VLSI Implementation of RSA Cryptosystem

A THESIS SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIRMENTS FOR THE DEGREE OF

MASTER OF TECHNOLOGY
in
Electronics and Communication Engineering
VLSI and Embedded System Design
by

SUSHREE RASMITA DASH
Roll No: 211EC2078

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
ROURKELA, ODISHA
INDIA
2013

VLSI Implementation of RSA Cryptosystem

A THESIS SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIRMENTS FOR THE DEGREE OF

MASTER OF TECHNOLOGY
in
Electronics and Communication Engineering
VLSI AND EMBEDDED SYSTEM DESIGN
by

SUSHREE RASMITA DASH
Roll No: 211EC2078

Under the guidance of
Prof. KAMALAKANTA MAHAPATRA

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
ROURKELA, ODISHA
INDIA
2013

Dedicated to My Nana & Bou

NATIONAL INSTITUTE OF TECHNOLOGY
Dept. of Electronics and Communication Engineering
Rourkela-769008, Odisha, India

CERTIFICATE

This is to certify that the work entitled in this thesis, "VLSI Implementation of RSA Cryptosystem**"** submitted by SUSHREE RASMITA DASH in partial fulfilment of the requirements for the award of Master of Technology Degree in **Electronics & Communication Engineering** with specialization in **VLSI and Embedded System Design** during 2011-2013 at the National Institute of Technology, Rourkela. This is an authentic work carried out by her under my supervision and guidance. To the best of my knowledge, neither this thesis nor any part of it has been submitted for any degree or diploma elsewhere.

Place: Rourkela **Dr. Kamalakanta Mahapatra**
Date: **(Professor)**

ACKNOWLEDGEMENTS

I am deeply indebted to **Prof. K. K. Mahapatra** of E&CE Department, supervisor of my project providing me the required guidance to complete the project successfully in time with his valuable support. He was always ready to share his knowledge at every stage of my project.

I sincerely thank to **Prof. D.P. Acharya**, **Prof A. K. Swain, Prof P. K. Tiwari,** and **Prof. N.M. Islam** for teaching and helping me during two year of M.Tech course. I would like humble thank to all the faculty members of Electronics and Communication Engineering Department for their help and guidance.

I would like to thank all my friends of VLSI specialization for immense support and my classmates for all discussions about study and project which make the project come to successful. I have enjoyed two year of M. Tech life at NIT, Rourkela with the companion of all my friends and Research scholars. I would to like express my hearty gratitude and special thanks to all the research scholars of VLSI Design and Embedded System specialization for their help during the research period for motivating and supporting in my project.

Lastly, I would like to thank my parents and well-wishers and expressing utmost gratitude before the Lord Jagannath.

Sushree Rasmita Dash

sushreedash@gmail.com

Abstract

In the age of information, security issues play a crucial role. Security comes with three points' confidentiality, integrity and availability. The entire above said thing will come from an efficient cryptographic algorithm. We need special hardware to implement these cryptographic algorithms to provide higher throughput. This hardware should have high flexibility since the cryptographic algorithms are constantly changing. To achieve this goal VLSI implementation of these cryptosystem is the best solution. Our work is mainly based on designing architectures for Rivest–Shamir–Adleman (RSA), one of the most well-known public key cryptosystem.

This work started with the understanding of basics of cryptography and modular arithmetic which is essential to understand cryptographic algorithm. Then in the cryptographic module RSA, a public key cryptographic module is chosen as the algorithm for implementation. The design goal is to increase the speed or in other words the throughput of RSA cryptosystem.

RSA operation is based on modular exponentiation of large integers. The size of the modulus is preserved at a minimum of 512 bits for security purpose. Modular exponentiation is a process in which repeated modular multiplication takes place. Since modular multiplication is a time consuming process, speed of RSA cryptosystem depends on the speed of modular multiplier and the number of modular multiplication performed. In this work Montgomery algorithm was imbibed to implement modular multiplication. For modular exponentiation MSB (most significant bit) first algorithm was chosen and Montgomery multiplier is used to do the exponentiation.

Montgomery multiplication replaces trial division with a series of additions and shifting operation, hence it comprises of adders of large operand size (at least 512 bit). So in this work we have studied different adder architectures. We have done a comparative study of modular multiplier by changing the adder structures used in it.

In adder structure the study began from the simple RCA (Ripple Carry Adder) structure. Then CSA (Carry Save Adder), CLA (Carry Look Ahead), CSL (Carry Select Adder), CSK (Carry Skip Adder) structures are studied and VHDL code was written for the above said structures and a comparative analysis has been done. From the analysis CSK and CSL structures are found to be fast in comparison to others. Then CSL structure is taken and it is modified to give better speed to be used in modular multiplier. Transistor level schematic of the modified CSL structure is simulated in Cadence tool to study the various parameters of the designed circuit.

The simulations presented in this work utilize the Xilinx ISE 10.1 Simulator environment. The Xilinx XST 10.1 was used in the synthesis of the implementation. For backend design Cadence tool was used.

Contents

List of Figures

List of Tables

List of Abbreviation

RSA	:	Rivest–Shamir–Adleman
RCA	:	Ripple Carry Adder
CSA	:	Carry Save Adder
CLA	:	Carry Look Ahead
CSA	:	Carry Select Adder
CSK	:	Carry Skip Adder
VHDL	:	Very High Speed Integrated Circuit Hardware Descriptive Language
MR	:	Montgomery Reduction
MP	:	Montgomery Product
MSB	:	Most Significant Bit
FSM	:	Finite State Machine
RTL	:	Register Transfer Level
PAR	:	Placement And Routing
PDP	:	Power Delay Product
LUT	:	Look Up Table
CMOS	:	Complementary Metal Oxide Semiconductor
LVS	:	Layout Versus Schematic
DRC	:	Design Rule Check
RC	:	Resistor Capacitor

Chapter 1 Introduction

1.1 Introduction

In the field of networking, role of network security is immense. In the age of information we need to keep information about every aspect of our live. These information needs to be hidden from unauthorized access (confidentiality), protected from unauthorized change (integrity), and available to an authorized entity when it is needed (availability). Hence the way of keeping the information securely is known as cryptography, [1] which comes from a word with Greek origin, means "secret writing". Many cryptographic algorithms are developed to achieve the above said goal. The algorithms should be such that an opponent cannot defeat its purpose. These algorithms generally consist of some arithmetic operations which are complicated and time consuming. It is because of the fact that these algorithms work with large amount of data either in blocks or simply in streams. Although a single traditional CPU is enough for performing these computations, but for a machine which works as a server in a huge network gets millions of client requests for performing cryptographic operations for them individually. This makes the workload huge. The computational resources may also be limited for example in smartcards, mobile phones, handheld computers, etc. Moreover if the associated network is of high speed, the speed of the necessary cryptographic computations also needs to be taken into account. For example in transmitting audio and video data for cable TV, video conferencing and sensitive financial and commercial data, the speed of the cryptographic module to be embedded, needs to be very high. So from the viewpoint of high speed and throughput, traditional software implementations of these complicated cryptographic algorithms are not efficient in real time applications like ATM, VPN, etc. This forces the system designers to go for hardware implementation of the cryptosystems.

Rivest–Shamir–Adleman (RSA) [3] cryptosystem is a well-known private key cryptosystem whose security comes from the fact that large integers are factorized inefficiently. In this thesis the goal is to design an efficient architecture for modular multiplication and exponentiation operations, which are the main operation of RSA algorithm.

1.2 Motivation and Objective

The demand for new network security systems is increasing with the growth of network services in our society. Cryptographic applications such as digital signature, e-commerce, ATM cards and watermarking, are very much computationally intensive and software approach for these applications is rather inefficient to work in line speed of the network. So the current trend is to replace such software applications with specialized hardware which are quite compatible to work in such high speed.

One of the most common public key algorithms in use for secure transaction is the RSA. Hence the goal to design an efficient architecture for RSA becomes more relevant. For high speed application we need to develop hardware to compute such mathematical operations which are involved in cryptographic algorithm.

One algorithm can be implemented in a variety of architectures, so by changing the architecture we can vary the flexibility of the algorithm.

Objective of this work is VLSI implementation of mathematical operation involved in RSA algorithm.

1.3 Literature Review

The mathematical concept of RSA algorithm was initially proposed by Diffie and Hellman in 1976 [2]. The application of the concept proposed by the above two was presented by Rivest–Shamir–Adleman in 1978[3], and the cryptosystem is known as RSA in the honor of the

above authors. The main operation of RSA algorithm is modular multiplication and exponentiation. Modular multiplication can be done in various ways [4]. The first way is multiply and reduces. In this way a parallel multiplier is there which multiplies two numbers then reducer is there which performs division operation. The second way is double add and reduce. In this method a modular adder is used to calculate the modular multiplication. The third way is Montgomery multiplication which is based Montgomery arithmetic proposed by P.L Montgomery [5] in 1987, which replaces the trial division by shifting and addition operation. Modular multiplier can be implemented in hardware in various ways by using Montgomery algorithm [6], [7], [8], [9], [10]. G. Sutar et al. [10] proposed architecture for Montgomery modular multiplier using digit serial computation process. In this process modular multiplier consists of numbers of adders. Study of different adder structure [11] concludes that CSL (Carry Select Adder) [12] structure gives best delay performance. For transistor level design, domino CMOS logic [13], [14] was taken as speed is more. All the blocks are coded using VHDL [15]. HDL codes were synthesized and simulated using Xilinx ISE10.1 tool [16].

1.4 Thesis Organization

This thesis is organized in such a way that structure of RSA cryptosystem can be understood by each computational block. It consists of seven chapters, and the different chapters are described as follows

Chapter 2 describes the basics of cryptography and introduction to RSA algorithm. A block diagram is presented to show different blocks of RSA cryptosystem.

Chapter 3 describes about modular multiplication operation. Implementation result for Montgomery modular multiplier is shown. How the structure of modular multiplier is modified is also described.

Chapter 4 concerns about modular exponentiation operation. It describes an algorithm which will create an exponentiation module. Synthesis and simulation result for this module is described.

Chapter 5 describes the study of different adder structure. A modified adder structure is also proposed and the results are discussed.

Chapter 6 describes the transistor level implementation of modified adder structure. Schematic of various blocks of adder was shown and simulation results are also given.

Chapter 7 summarizes the whole work and contribution to achieve the goal of the work is described. Extension of this work is also presented.

Chapter 2 Basics of Cryptography and RSA Algorithm

2.1 Introduction

This chapter deals with the discussion of basics of cryptography and RSA (Rivest–Shamir–Adleman) algorithm, which is a public key or asymmetric key cryptosystem. This chapter consists of three parts. In the first part basics of cryptography have been discussed. In the second part basics of asymmetric key cryptosystem have been discussed. In the final part detail of RSA algorithm have been discussed.

2.2 Basics of Cryptography

Cryptography is the study of building ciphers to ensure that the information are hidden from unauthorized access (confidentiality) and protected from unauthorized changes (integrity). In other words it is the process of converting the plaintext into ciphertext and vice versa. The above things are done by using some cryptographic algorithm and secret keys. The original message that a sender is going to send is known as plaintext and the message that is sent through the channel is known as ciphertext. In classical cryptography, the same secret key is used for encryption and decryption known as symmetric key cryptography. On the other hand in modern cryptography different secret keys are used for encryption and decryption known as asymmetric key cryptography.

2.3 Asymmetric Key Cryptosystem

In asymmetric key cryptography Encryption and decryption is carried out by using two different keys. The two keys are referred to as the public key and the private key, hence otherwise known as public key cryptography. Sender A encrypts the message by using receiver B's public key to provide confidentiality. This enciphered message can be deciphered by B only by the private key of B. To send an authenticated message to B, A has to encrypt the message with his own private key. This message can only be decipherable with the help of public key of A. In our work different hardware are designed and implemented in

FPGA used in RSA algorithm, an asymmetric key cryptographic algorithm. The main aim of the work is to increase the speed of the individual computational block used in RSA cryptosystem.

2.3.1 Mathematics of Asymmetric Key Cryptography

In the design of asymmetric key cryptography number of concepts of number theory is required to know. This section discusses about some of them.

2.3.1.1 Prime Number

An integer p>1 is a prime number if and only if it has exactly two divisors 1 and itself i.e. p.

Two integers p and q are said to be coprime or relatively prime if gcd (p,q) = 1.

2.3.1.2 Euler's Totient Function (φ (n)):

It is defined as the number of positive integers less than n and relatively prime to n. By convention, φ (1) = 1.

2.3.1.3 Fermat's Theorem:

Fermat's theorem states the following: If pis prime and a is a positive integer not divisible by p, then

ap-1 ≡ 1 (mod p).

This theorem gets modified to: if p is a prime and a is an integer, then

ap ≡ a (mod p).

This theorem is helpful for quickly finding a solution to some exponentiation. This theorem also helps in finding multiplicative inverses quickly if the modulus is a prime. If p is a prime and a is an integer such that p does not divide a, then a-1 mod p = ap-2 mod p.

2.3.1.4 Euler's Theorem

Euler's theorem states that for every a and n that are relatively prime:

a φ (n) ≡ 1 (mod p).

2.3.2 RSA Cryptosystem

This algorithm is proposed by Rivest, Shamir & Adleman of MIT in 1977. It is the best known & widely used public-key scheme whose security comes from the fact that large integers are factorized inefficiently. It is based on modular exponentiation over large integers. Two keys are generated one is public and the other is private. The size of the modulus is at least 512 bits for security purpose. If n users are there n no. of private key is generated.

RSA uses two algebraic structures 1) a public ring R = <Zn, +, ×>, 2) a private group G = <Z φ (n)*, ×>. Key generation can be done in the following way:

Two random prime numbers p, q is taken.

System modulus n = pq is calculated.

Euler's totient φ (n) = (p-1) (q-1) is calculated.

A random encryption key is selected such that 1<e< φ (n) and gcd (e, φ (n))=1.

Decryption key d is calculated such that d = e-1mod φ (n).

Publish the public key {e,n}.

Keep the private key {p,q,d}.

To encrypt a message M a sender uses the public key of the recipient and compute the cipher text C as follows:

$$C = Me \bmod n$$

To decrypt a message C the recipient uses his own private key and computes the plain text M as follows:

$$M = Cd \bmod n.$$

2.3.2.1 RSA Structure:

RSA structure contains following module

Key generation module

Encryption module

Decryption module

Key Generation Module:

This module generates the private and public key for the algorithm. First it will take two prime number p and q, a multiplier will be there two multiply these numbers to compute the modulus value n. then Euler's totient function is computed with the help of subtractor and multiplier. One GCD calculator is there to check weather φ (n) and e are relatively prime or not. Then a multiplicative inverse calculator is there to calculate d, the private key. Block diagram of key generation module is shown in Fig. 2.1.

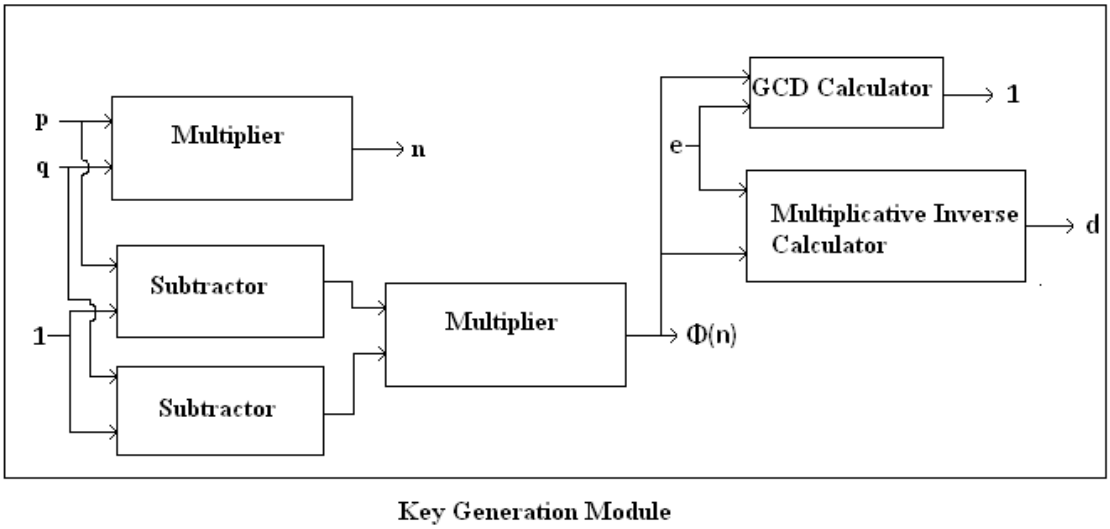

Fig. 2.1 Block diagram of Key Generation Module of RSA cryptosystem.

Encryption Module:

This module performs encryption operation which is nothing but a modular exponentiation. So the main computational block is a modular exponentiation which consists of a modular multiplier, some registers and a FSM. The detail structure of this block is described in chapter4. The block diagram of this module is shown in Fig. 2.2.

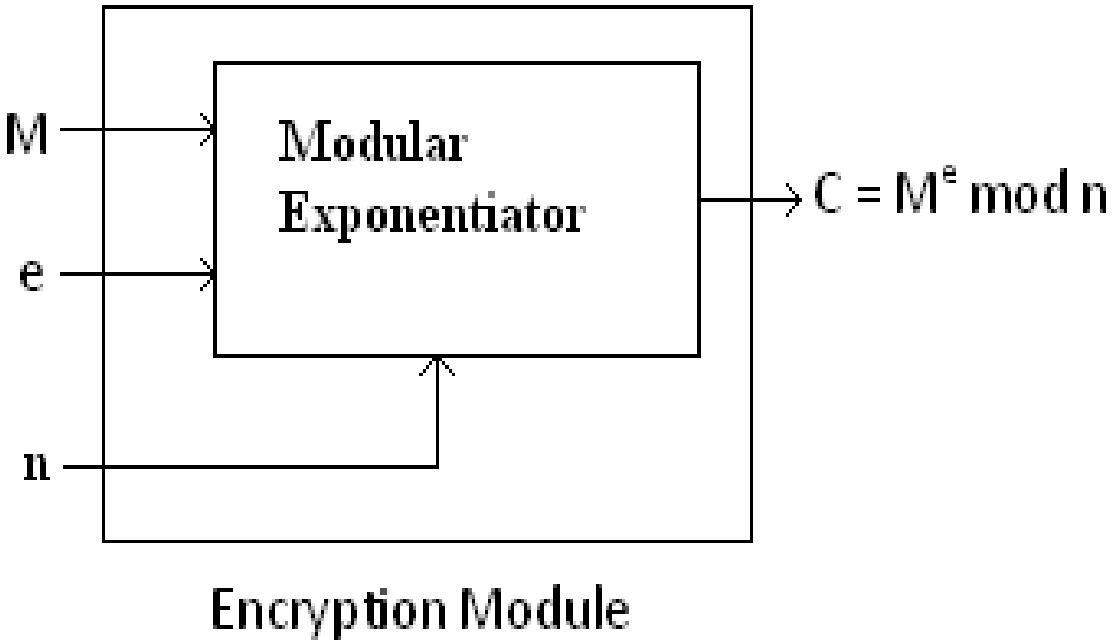

Fig. 2.2 Encryption Module of RSA cryptosystem.

Decryption Module:

This module is same as the encryption module as both the encryption and decryption algorithm consists of same mathematical operation i.e. modular exponentiation. The block diagram of decryption module is shown in fig.2.3.

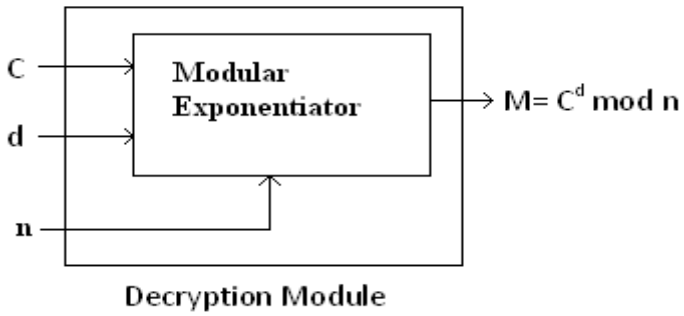

Fig. 2.3 Decryption Module of RSA cryptosystem.

2.3.2.2 RSA Example

Select two prime numbers, p= 17 and q= 11.

Calculate n= pq= 17 x 11 = 187.

Calculate φ(n) = (p1)(q1) = 16 x 10 = 160.

Select esuch that eis relatively prime to φ(n) = 160 and less than φ(n) we choose e= 7.

Determine d such that d ≡ 1 (mod 160) and d< 160. The correct value is d= 23, because 23 x 7 = 161 = 10 x 160 + 1.

Suppose M = 88, then C = 887 mod 187 = 11.

We can get back M by decrypting C as follows:

M = 1123 mod 187 = 88.

2.4 Conclusions

This chapter describes the basics of cryptography and RSA algorithm in detail. Some mathematical concept that is important for design of asymmetric key cryptosystem has also been discussed. From this chapter we come to know about the computational blocks that are present in RSA cryptosystem. In our work only modular multiplication and exponentiation block have been designed using VHDL, an HDL and implemented using Xilinx ISE10.1 software. In the next chapter individual computational blocks are described in detail. Simulation, synthesis and implementation results have been described for each computational block.

Chapter 3 Modular Multiplication Hardware

3.1 Introduction

As we have seen in chapter two, the RSA algorithm is based on modular exponentiation operation. We need to design a hardware that is able to perform the modular exponentiation. This modular exponentiation can be done by repeated modular multiplication. So for analyzing the modular exponentiation, we need to go through modular multiplication. This chapter describes the theory and algorithm for computing modular multiplication as well as the synthesis and simulation results for the same. In this chapter we will study algorithm for computing

$$z = x.y \pmod{m},$$

Where z, x, y and n are k bit binary number and x, y € Z_m. Since k is often more than 256 bit, we need to build an architecture in order to deal with these large numbers. In this work we focus on the number of ways of computing modular multiplication. Then we will discuss about Mongomery algorithm for computing modular multiplication in detail. Later synthesis and simulation result for Montgomery modular multiplication block has been discussed.

3.2 Modular Multiplication

Given x, y € Zm = {0, 1, 2… m-1}, w have to compute z = xy mod(m), where x, y, z are k bit natural number. Modular multiplication can be done in various ways: 1) Multiply and reduce: this is the straight forward method in which two numbers are multiplied together and then the value is reduced to n bit with the help of a reducer circuit. So th hardware consists of any type of multilplier and any reducer. 2) Double add and reduce: in this way modular multiplication is carried out by addition, doubling and then reduction operation. Hardware of this algorithm consists of a modular adder. 3) Montgomery production: in this way modular multiplication is based on an algorithm P.L. Montgomery, which replaces the trial division method using addition and shifting operation.

3.2.1 Montgomery Modular Multiplication

In Montgomery's algorithm the integers are transformed into m residues and multiplications are performed with these m residues. First all the integers are converted into Montgomery domain with the help of Montgomery reduction algorithm, which can be described as follows.

Consider two natural numbers m, R > m and both are relatively prime to each other, that is, gcd (m,R) = 1. Then there exist an element R-1 of Zm such that

RR-1 mod m = 1.

For any given natural number x, Montgomery reduction MR is defined as:

MR (x) = xR-1 mod m.

As gcd of m, R is 1, there exists an element –m-1 of ZR such that m (–m-1) mod R = -1. Suppose the value of –m-1 is previously computed and x < mR, if q ≡ x –m-1 (mod R), then z = (x+qM)/R is an integer and further,

z ≡ xR-1 mod m.

For any two natural numbers x, y Montgomery reduction is defined as:

MR (xy) = xyR-1 mod m.

Given x and y in Zm, their product p = xy is smaller than mm < mR, so that their Montgomery product can be computed as follows:

MP (xy) = MR (xy).

So in Montgomery multiplication Montgomery reduction step is performed. This algorithm can be converted into binary format when we are taking R as a power of 2. Steps involved in Montgomery algorithm in binary format is shown below:

Let x, y are two k bit binary number, then x (i) € {0, 1} represents the ith bit of x; therefore x = ∑ k-1i=0 x(i) × 2i.

Algorithm 3.1: Binary Montgomery Multiplication Algorithm

p := 0;

for i in 0 to k-1 loop

a:= p + x(i) * y; 3.1

if (a mod 2) = 0 then p=a/2;

else p:= (a+m)/2; end if;

end loop;

if p >= m then z := p-m; else z := p; end if;

p is the intermediate product term.

In the above algorithm LSB of equation 3.1 can be found out in a simpler way. If x(i) = 1 then LSB of (1) is LSB of p and y. Else it is the LSB of p. So new value of p can be computed as p = p + x(i) × y + q(i) × m. so the algorithm can be modified as follows:

Algorithm 3.2: Modified Binary Montgomery Multiplier Algorithm

p := 0;

for i in 0 to k-1 loop

q(i) := (p(0) + x(i) * y(0)) mod 2;

p := (p + x(i)*y + q(i) * m)/2;

end loop;

if p >= m then z := p-m; else z := p; end if;

3.2.2 Architecture of Montgomery Modular Multiplier

Architecture corresponding to Algorithm 3.1 is shown in Fig. 3.1. Shift register will give one bit of x at a time.

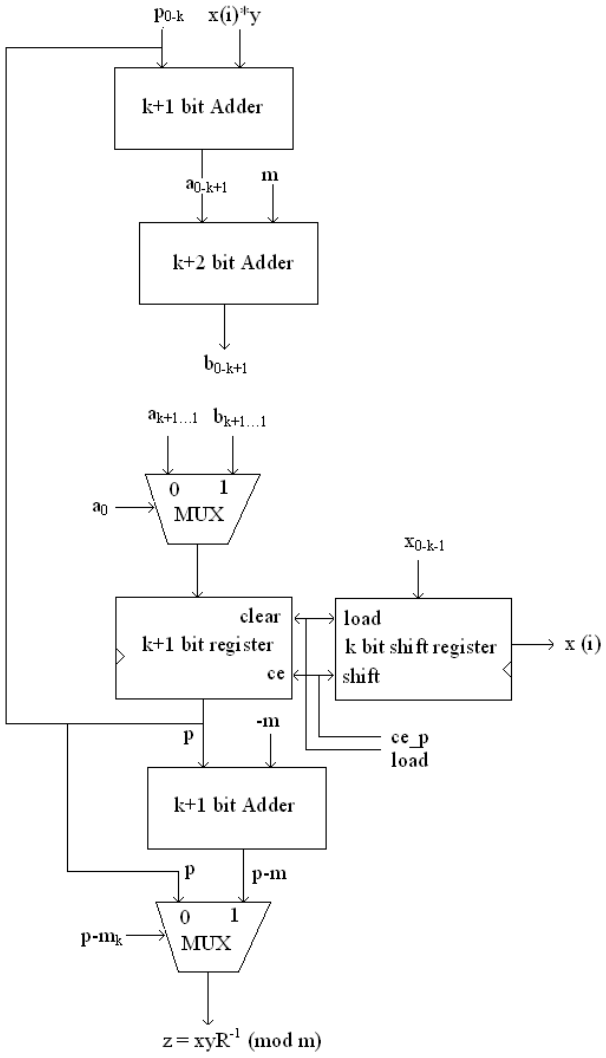

Fig. 3.1 Architecture for algorithm 3.1

p is initialized to zero by k+1 bit register and it updates the value of p in each clock cycle. Adders are there to compute a, b, p-m and multiplexers are there for shift operation and output calculation. Structure for algorithm 3.2 is shown in Fig. 3.2, in which first multiplexer is not there.

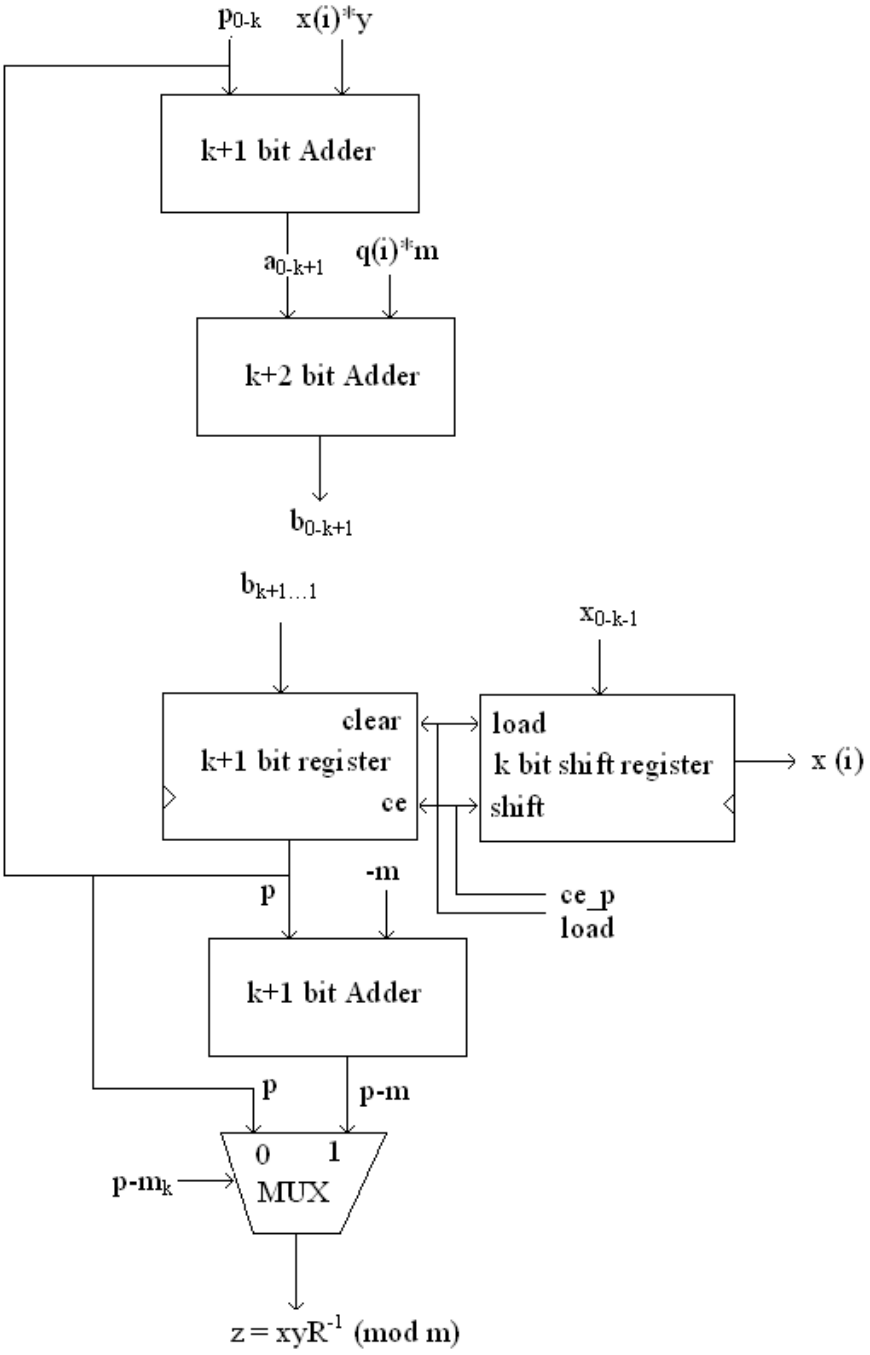

Fig. 3.2 Architecture for Algorithm 3.2

In the above two structures the critical path consists of computation of p and q (i). So to improve the speed of the multiplier we need to reduce the critical path, which can be done by using faster adder structure. In our study a comparative analysis is done by taking different adder structure to see how it affects the speed of the design.

3.3 Montgomery Modular Multiplier Implementation

The above architectures that were described in 3.2.2 were coded using VHDL and synthesized and simulated using Xilinx ISE 10.1 tool. Synthesis and simulation result is shown in the next sub sections.

3.3.1 Simulation Results

Simulation for a Mongomery modular multiplier with modified adder structure is shown in the figure given below.

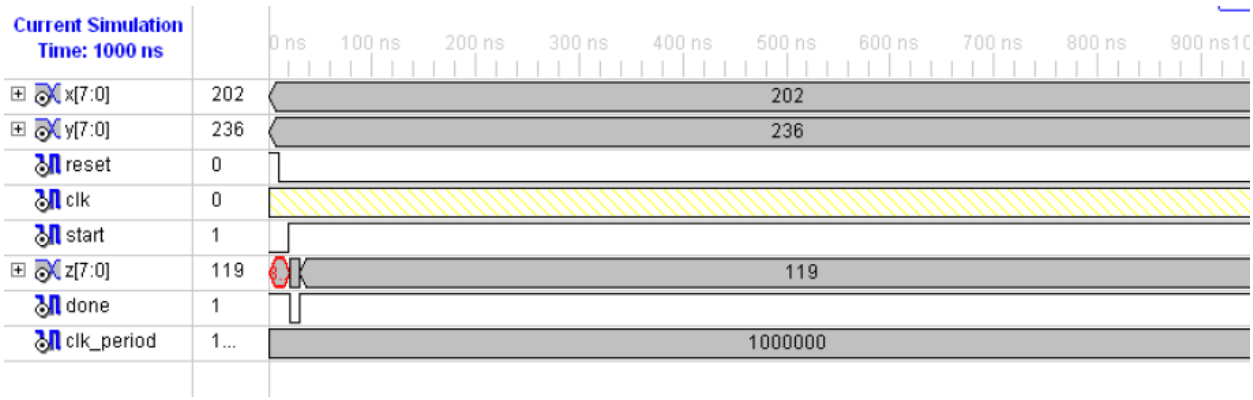

Fig. 3.3 Simulation result for Montgomery modular multiplier

For verification purpose x and y are taken as 8 bit number.

m is initialized to 239

R is taken to be 256, so R-1 (mod m) is found to be 225.

x is taken as 202

y is taken as 236.

Hence xy (mod m) = xyR-1 (mod m) = 202.236.225 (mod 239) = 119.

3.3.2 Synthesis Results

Synthesis results for Algorithm 3.1 and 3.2 with RCA, CSA and modified adder are shown in tabular formats, which are given below:

Table 3-1 Device utilized for 512 bit Montgomery modular multiplier in FPGA

Algorithms	No. of flip flops used	No. of slices used
Algo. 1 using RCA	1064	3593
Algo. 2 using CSA	1042	2596
Algo. 1 using modified CSL	1048	4356
Algo. 1 using modified CSL	1048	4591

Table 3-2 Timing report for 512 bit Montgomery modular multiplier.

Algorithms	Total time in ns	Operating Frequency in MHz	Throughput in Mbps
Algo. 1 using RCA	41.970	35.9	12.199
Algo. 2 using CSA	41.302	27.806	12.396
Algo. 1 using modified CSL	12.750	78.432	40.156
Algo. 1 using modified CSL	12.646	79.075	40.487

For synthesis process virtex5 FPGA is taken. Throughput of the above design can be calculated as No. of bits/ Timing. From the report it is shown that multiplier with modified adder structure gives better throughput.

Study of RTL schematic for the above algorithm found that architectures for multiplier using modified adder infer 1024 flip flops to create registers which store the input value. The above hardware takes 512 clock cycles to calculate the final result.

3.4 Conclusions

In this chapter modular multiplier block of RSA cryptosystem was described in detail. Montgomery algorithm was taken to calculate modular multiplication of two integer value. The design was coded using VHDL, an HDL and synthesized and simulated using XILINX ISE 10.1 software. Adder block present in modular multiplier is changed with a modified adder structure which will be explained in chapter 5. Synthesis and simulation result for different modular multiplier module was shown in a tabular format and the result was analyzed. In the next chapter modular exponentiation block is described

Chapter 4 Modular Exponentiation Hardware

4.1 Introduction

Modular exponentiation is a process in which repeated modular multiplication takes place. This chapter describes the algorithm, architecture of modular exponentiation process. Synthesis and simulation result for the above said operation is also described in this chapter. In this chapter we will describe the algorithm for computing

$$Z = y^x \pmod m$$

Where z, x, y and n are k bit binary number and x, y € Z_m = {0, 1, 2... m-1}. In this chapter we will discuss about the architecture of the modular exponentiation algorithm. Synthesis and simulation result of this algorithm is shown.

4.2 Modular Exponentiation

It is a process in which we have to compute z = yx (mod m), where x, y, z are k bit natural number. This can be done in various ways [5], such as MSB first, LSB first, m – ary method etc. in the first method most significant bit of x is processed first, in the second method least significant bit of x is processed first, and in third method m no. of bit of x is processed at a time. In this work modular exponentiation architecture is based on MSB first algorithm.

4.2.1 Modular Exponentiation Algorithm

Given x and y belonging to Zm = {0,1,…., m-1}, compute z = yx (mod m) .Assume that m is a k bit number and that x is represented in base 2, that is x = xk-1.2k-1 + xk-2.2k-2 +……+ x0.20. Then z can be written in the form [6]

z = ((….(12yxk-1)2)yxk-2)2….)2yx1)2yx0 (mod m)

The above equation corresponds to following algorithm, which is known as MSB (most significant bit) first algorithm, as the MSB of x is first processed.

Algorithm 4.1:

```
e := 1;
for i in 1 to k loop
        e = (e*e) mod m;
if x(k-i) = 1 then e = (e*y) mod m; end if;
end loop;
z := e;
```

When we are replacing the modular multiplication process with Montgomery modular multiplication then initialization of e is done with 2k (mod m) and y with ty = MP(y,22k (mod m)). Mod m products are replaced with Montgomery product. The modified MSB first exponentiation algorithm using Montgomery modular multiplication is given below:

Algorithm 4.2:

```
e := exp_k;
ty := mp(y, exp_2k);
for i in 1 to k loop
e = mp (e,e);
if x(k-i) = 1 then e = mp (e,y) ; end if;
```

end loop;

z := mp (e,1);

Where exp_k = 2k mod m and exp_2k = 22k mod m and they are previously computed.

4.2.2 Architecture of Modular Exponentiation Block

As we know modular exponentiation can be done by repeated modular multiplication, architecture of modular exponentiation circuit consists of modular multiplier, some parallel registers to store the intermediate values, a parallel in serial out register to scan one bit of x at a time, and multiplexers to choose the value of the operand of modular multiplier. An FSM is designed to control all these components. Fig. 4.1 shows the architecture for modular exponentiation circuit using MSB first algorithm.

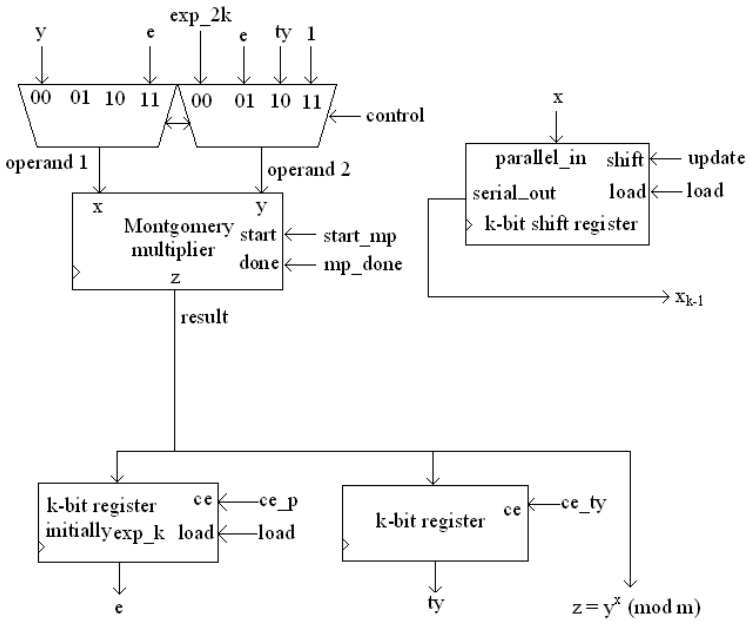

Fig. 4.1 Architecture for Algorithm 4.2.

4.3 Modular Exponentiation Implementation

The above architectures that were described in 4.2.2 were coded using VHDL and synthesized and simulated using Xilinx ISE 10.1 tool. Synthesis and simulation result is shown in the next sub sections.

4.3.1 Simulation Results

Simulation result for modular exponentiation which follows MSB first algorithm is shown in Fig. 4.2.

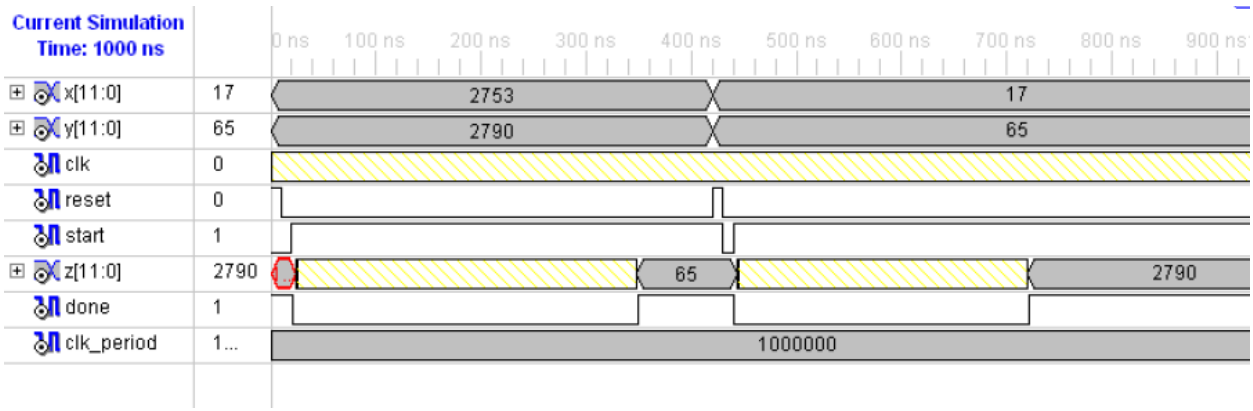

Fig. 4.2 Simulation result for modular exponentiation block.

p was taken as 61, q was taken as 53.

So m was initialized to 3233.

exp_k = 2k (mod m) was precomputed and initialized to (35F)16.

exp_2k = 22k (mod m) was precomputed and initialized to (49B)16.

e (public key) was taken as (17)10, d (private key) was found to be (2753)10.

Plaintext was initialized as y = (65)10.

We get ciphertext as C = z = 2790 = Me (mod m) = 6517 (mod 3233).

Similarly from ciphertext C we get the plaintext = M = z = 65 = Cd (mod m) = 27902753 (mod 3233).

4.3.2 Synthesis Results

No of slices used : 3098/69120

No of LUTs used : 4655/69120

Timing Report :

Minimum period : 17.259ns

Maximum Frequency:57.259MHz

Min. i/p arrival time: 2.067ns

Max. o/p required time : 20.058ns

From the above parameter the throughput of the exponentiation block was found to be 29.665Mbps. The RTL schematic diagram of the exponentiation block is shown in Fig. 4.3.

Fig. 4.3 RTL schematic of modular exponentiation block.

4.4 Conclusions

In this chapter modular exponentiation block of RSA cryptosystem was described in detail. MSB (Most Significant Bit) first algorithm was taken to calculate modular multiplication of two integer value. The design was coded using VHDL, an HDL and synthesized and simulated using XILINX ISE 10.1 software. Modular multiplier block present in modular exponentiation block is changed with our modular multiplier block. Synthesis and simulation result for modular exponentiation module was shown and the result was analyzed. In the next chapter different adder structures are described.

Chapter 5 Study of Different Adder Structures

5.1 Introduction

Adders are basic blocks of Montgomery multiplier as we have discussed in chapter 3. As the operand sizes are large for the adders, they need to be fast. Speed of the adder depends on the carry propagation process. Various adder structures [7] have been implemented to make it fast. In this work we have taken RCA (Ripple Carry Adder), CLA (Carry Look-ahead Adder), CSK (Carry Skip Adder) and CSL (Carry Select Adder) adder structures for comparative analysis. VHDL, an HDL is used to model these adder structures and then simulated it using Xilinx ISE 10.1. Among these four it is found that CLA and CSK to be faster. Then CSL adder structure is modified to get better speed, which is discussed in this chapter.

5.2 Different Adder Structures

5.2.1 Ripple Carry Adder (RCA)

In this adder full adders are connected in series, with the carry output from each full adder connected to the carry input of the next full adder in a cascaded manner. Carry bits are rippled from one bit to the next.

The two Boolean functions for the sum and carry are:

Sum = Ai xor Bi xor Cin

Cout = Ai . Bi + (Ai xor Bi).Cin

Where Ai, Bi are the operands of i bit length.

Cin is the input carry bit.

Since carry bits are rippled from one stage to the next, propagation delay increases for RCA structure as the size of the operand increases. Block diagram of RCA is shown in Fig. 5.1.

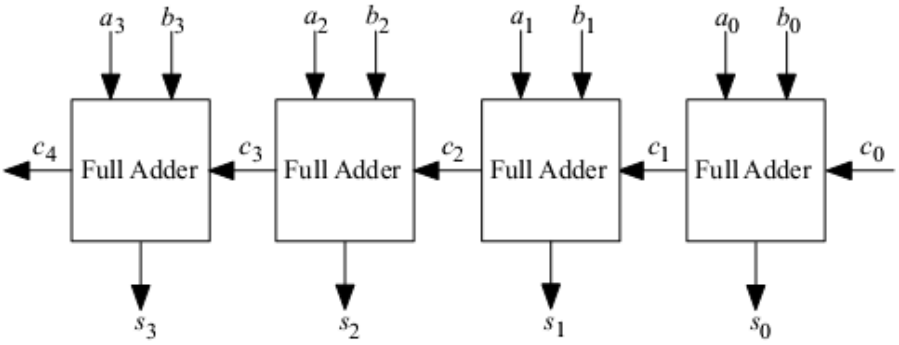

Fig. 5.1 Block diagram of RCA.

5.2.2 Carry Look-ahead Adder (CLA)

Carry propagation delay of RCA is solved in CLA by calculating the carry signals in each stage based on the input carry signal. A carry is generated when both Ai and Bi are 1. A carry is propagated when one of the two above bit is 1.

The Boolean equation for carry generate (Gi) and carry propagate (Pi) signals are:

Pi = Ai xor Bi

Gi = Ai . Bi

For a 4 bit CLA, a carry signal for each stage can be calculated from the following Boolean equations:

C1 = G0 + P0.Cin

C2 = G1 + P1.G0 + P1.P0.Cin

C3 = G2 + P2.G1 + P2.P1.G0 + P2.P1.P0.Cin

C4 = G3 + P3.G2 + P3.P2.G1 + P3.P2.P1.G0 + P3.P2.P1.P0.Cin

Boolean equation for the sum is:

Si = Pi xor Ci

But the disadvantage of CLA is that carry logic block gets complicated for more than 4 bits. For that reason to design higher bit CLA 4 bit CLAs are used in hierarchical structure. Block diagram of CLA structure is shown in Fig. 8.

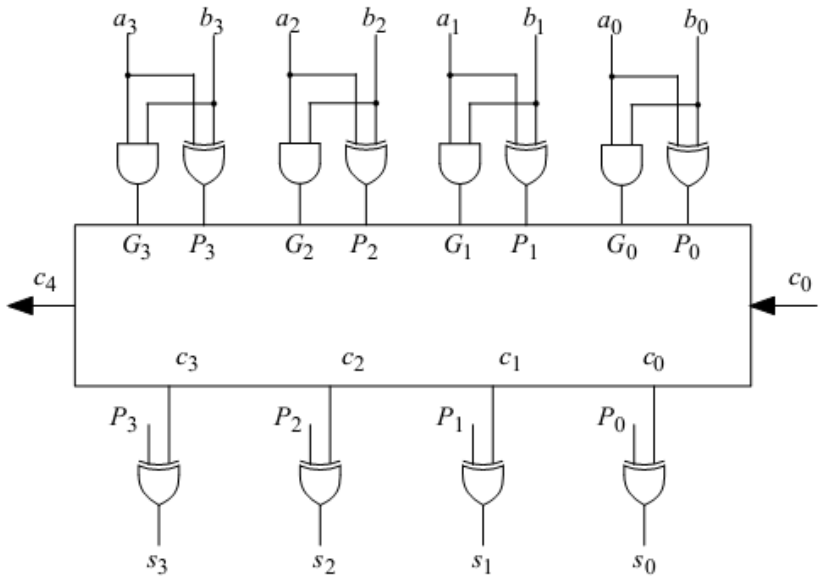

Fig. 5.2 Block diagram of CLA structure.

5.2.3 Carry Skip Adder (CSK)

In this adder structure entire block is divided into multiple blocks of 4 bits RCA structure. In carry skip logic, for every block a block carry propagate signal and a block carry out signal is generated. This logic is used in between two consecutive blocks.

Boolean equations for block carry propagate (P[4k, 4k+3]) and block carry out (CB(k + 1)) signal are:

P[4k, 4k+3] = P4k . P4k + 1 . P4k + 2 . P4k + 3 where k = no. of block

CB(k + 1) = C4k + 4 + P[4k, 4k+3] . CB(k).

Block diagram of CSK structure is shown in Fig. 5.3 and Fig. 5.4.

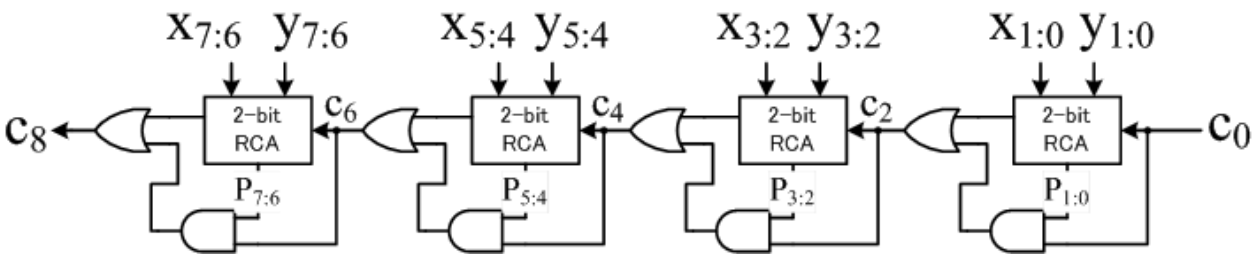

Fig. 5.3 8 bit CSL using 2 bit RCA.

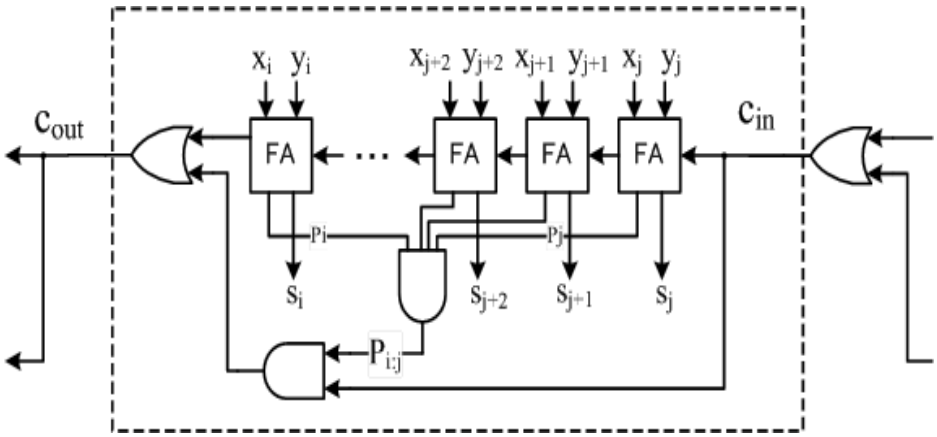

Fig. 5.4 Carry skip block.

5.2.4 Carry Select Adder (CSL)

In this adder Ai and Bi are divided into k blocks possibly of different size. First one 4 bit RCA is implemented by taking C0 as Cin , then rest of the blocks are generated by taking Cin both 1 as well as 0. Depending on the value of Cout of 1st RCA, sum and carry bit of next blocks are calculated using multiplexers.

This adder occupies a larger area but at the same time it gives shorter delay. Block diagram of CSL is shown in Fig. 5.5.

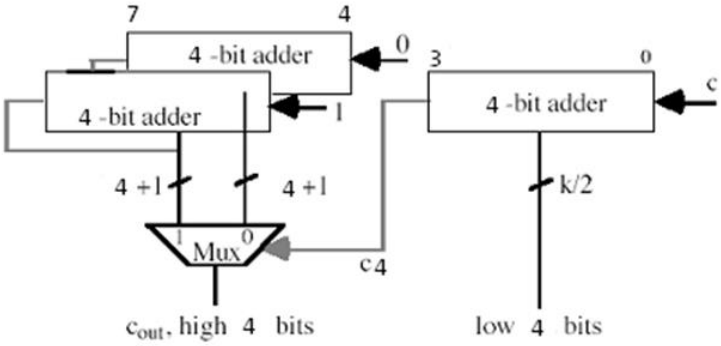

Fig. 5.5 Block diagram of CSL structure.

5.2.5 Modified Adder Structure

CSL [8] provides a substantial compromise between the RCA, which occupies a small area and has a longer delay, and the CLA, which occupies a lrger area and has a shorter delay [9]. In our proposed CLA both n bit operands, Ai and Bi are divide into k blocks of 4 bit sizes. The first block is a CLA by taking Cin = '0'. From the second block onward two additions are executed in parallel one by taking Cin = '0' and other by taking Cin = '1'. These blocks are also 4 bit CLA blocks. So for second blocks onward two output carry and two sum signals are generated and one of sum and carryout signal is selected by using multiplexers by taking carry out signal of previous block as a selection signal. Block diagram of a 8 bit modified adder structure is shown in **Fig. 5.6**.

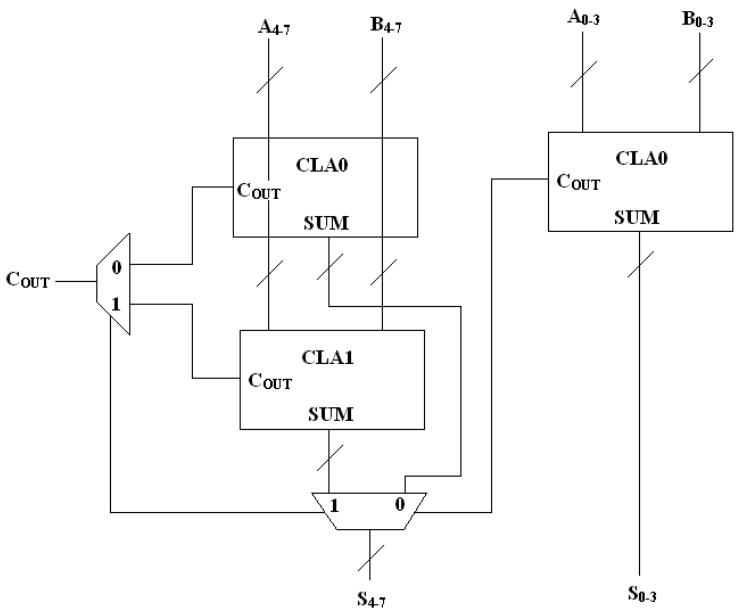

Fig. 5.6 Block diagram of Modified CSL structure.

Boolean expression of each signal present in CLA0 structure is given below:

Pi = Ai xor Bi

Gi = Ai and Bi

C1 = G0

C2 = G1 or (P1 and G0)

C3 = G2 or P2 (G1 or (P1 and G0))

COUT = G3 or P3 (G2 or P2 (G1 or (P1 and G0)))

Si = Pi xor Ci

Boolean expression of each signal present in CLA1 structure is given below:

Pi = Ai xor Bi

Gi = Ai and Bi

C1 = G0 or P0

C2 = G1 or P1 and (G0 or P0)

C3 = G2 or P2 (G1 or P1 and (G0 or P0))

COUT = G3 or P3 (G2 or P2 (G1 or P1 and (G0 or P0)))

Si = Pi xor Ci

Where Ai and Bi are two i bit operands, Pi is the carry propagate term, Gi is the carry generate term, C1-3 are the internal generated carry term, COUT is the output carry which will go to the next block.

5.3 Implementation of Different Adder Structures

Various adder structures those are described in 5.2 are coded using VHDL, simulated and synthesized in Xilinx environment. Synthesis and simulation results are shown in the next sub sections.

5.3.1 Simulation results

Simulation result for different adder structures are shown in Fig. 5.6 to 5.11.

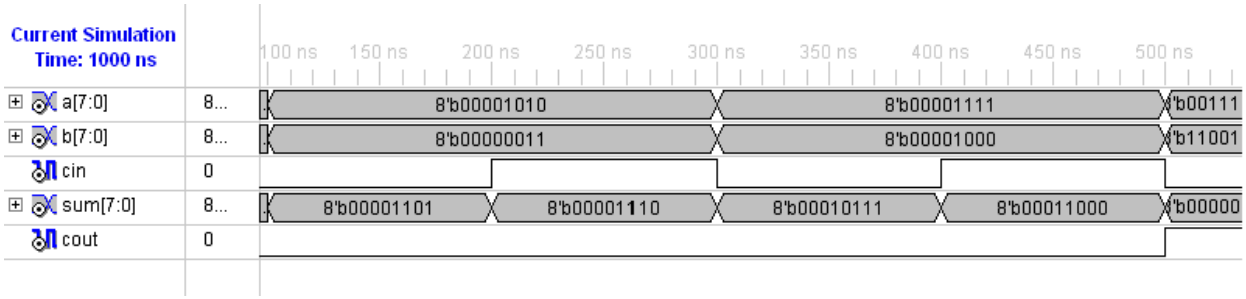

Fig. 5.7 Simulation result of RCA structure.

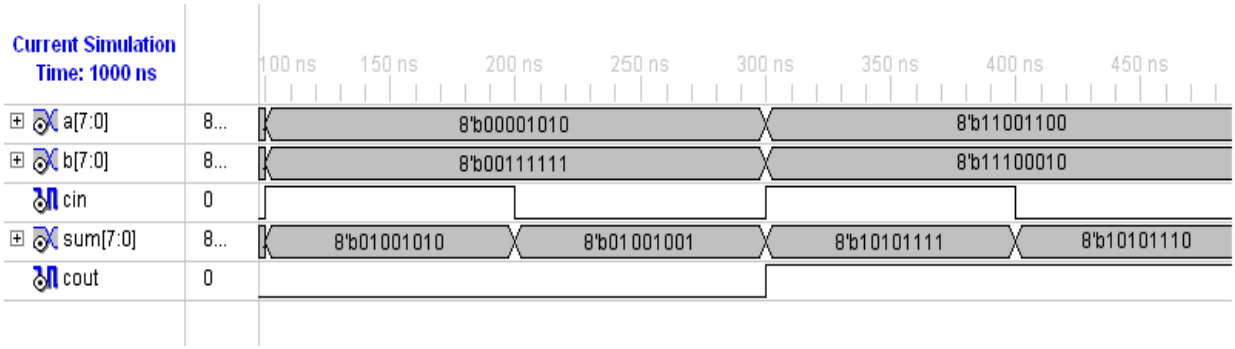

Fig. 5.7: Simulation result of CLA structure.

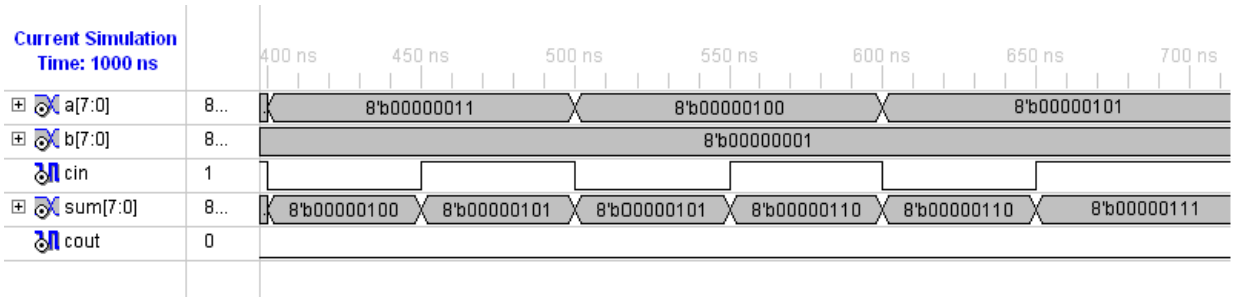

Fig. 5.8 Simulation result of CSK structure.

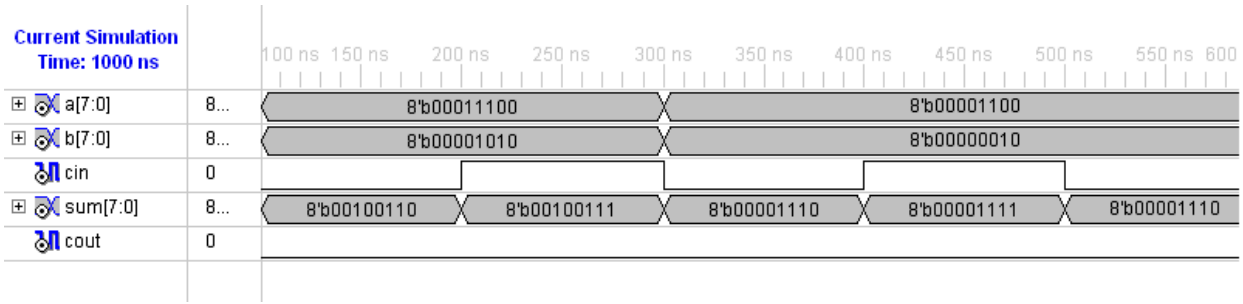

Fig. 5.9 Simulation result of CSL structure.

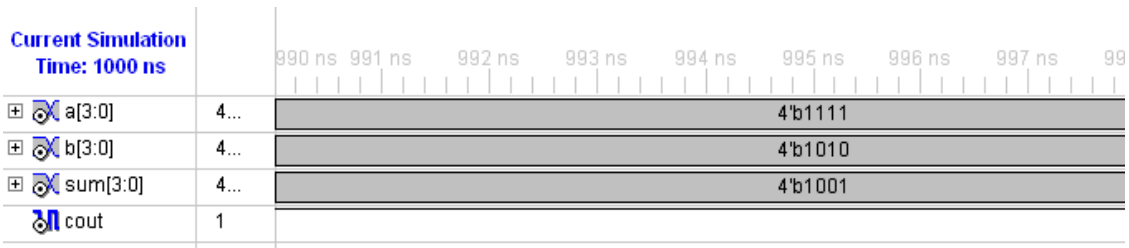

Fig. 5.10 Simulation result of modified CLA structure taking Cin = '0'.

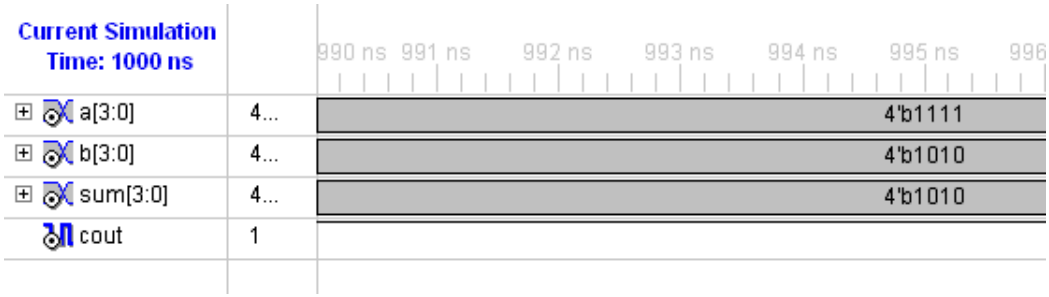

Fig. 5.11 Simulation result of modified CLA structure taking Cin = '1'.

5.3.2 Synthesis Results

Calculation of delay for different adder structure and different bit width and the values are shown in Table 5.1.

Table 5-1 Propagation delay of different adder structure for various bit size

Adders	Delay in ns						
	4 bit	8 bit	16 bit	32 bit	64 bit	128 bit	256 bit
RCA	9.926	14.846	24.686	44.366	83.726	162.446	319.886
CLA	9.882	13.612	21.072	35.992	65.832	125.512	244.872
CSK	10.088	15.170	20.962	35.706	66.406	125.382	242.122
CSL	10.006	13.654	21.110	36.022	65.846	125.494	244.790

Calculation of area, power and power delay product (PDP) for different adder structures of different bit size is done in synopsis and the values are shown in Table 5.2, 5.3 and 5.4.

Table 5-2 Area of different adder structures for various bit size

Adders	Area in μm^2						
	4 bit	8 bit	16 bit	32 bit	64 bit	128 bit	256 bit
RCA	40.32	80.639	161.279	322.559	645.119	1290.239	2580.479
CLA	39.96	78.48	159.8399	319.679	639.359	1278.719	2557.439
CSK	39.96	78.48	159.8399	319.679	639.359	1278.719	2557.439
CSL	96.84	136.79	329.759	727.919	1514.159	3086.369	6231.599

Table 5-3 Total dynamic power of different adder structures for various bit size

Adders	Total Dynamic Power in µw						
	4 bit	8 bit	16 bit	32 bit	64 bit	128 bit	256 bit
RCA	8.2595	17.0831	34.9638	70.7338	141.0575	283.4448	567.7568
CLA	7.7205	15.1787	32.7059	66.1703	132.1847	265.5256	531.9682
CSK	7.7205	15.1787	32.7059	66.1703	132.1847	265.5256	531.9682
CSL	17.2144	25.9197	61.8380	140.9388	291.64	595.2087	1202.1

Table 5-4 Power delay product (PDP) of different adder structures for various bit size

Adders	Power Delay Product in 10^{-15} w-s						
	4 bit	8 bit	16 bit	32 bit	64 bit	128 bit	256 bit
RCA	81.983	253.615	863.116	3138.175	11810.18	46044.473	181617.45
CLA	76.293	206.612	689.178	2381.601	8701.983	33326.649	130264.117
CSK	77.884	230.260	685.581	2362.676	8777.857	33292.130	128801.204
CSL	172.247	353.907	1305.4	5076.897	19203.327	74695.120	294262.059

Table 5.4: Power delay product (PDP) of different adder structures for various bit size.

Comparison of CSL structure versus modified adder structure is shown below:

CSL with CLA (32 bit)

Before PAR:

No of slices used: 50/4656

No of LUTs used: 93/9312

Delay: 26.591ns

After PAR:

No of slices used: 53/4656

Delay: 30.572ns

CSL with RCA (32 bit)

Before PAR:

46/4656

85/9312

27.573ns

After PAR:

48/4656

31.449ns

For 128 bit:	For 128 bit:
Delay: 44.259ns	86.485ns
For 512 bit:	For 512 bit:
Delay: 66.343ns	330.517ns

From the above result it is clarified that modified adder structure gives best result for delay when bit size increases.

5.4 Conclusions

In this chapter different adder structures are studied. A modified adder structure is proposed and described. Simulation and synthesis results for different adder structure as well as modified adder structure are shown in tabular format. Comparative analysis of modified adder structure and CSL structure is described. From the result it was found that modified adder structure gives improved delay performance as compared to other adder structure when bit size if the operand increases. In the next chapter transistor level design of modified adder structure is discussed.

Chapter 6 Transistor Level Design of Modified Adder Structure

6.1 Introduction

Transistor level design of modified adder structure, which has been discussed in last chapter, is described in this chapter. The design has been done by using domino CMOS logic, which is a combination of dynamic CMOS logic and CMOS inverter stage. Modified CLA circuit by taking C_{in} as 0 as well as 1 is designed in schematic level in 90nm technology. Transistors are taken from gpdk library. Bottom level blocks such as inverter, buffer, 2 input XOR, 2 input AND and 2 input OR gates are drawn in schematic level, simulated and then layout is drawn. After that LVS, DRC, RC extraction is done followed by post layout simulation. In the further sections of this chapter basic principle of domino CMOS logic style is discussed. Schematic diagram and simulation results of CLA circuits are shown and analyzed.

6.2 Basic Principle of Domino CMOS Logic

Combination of n – type dynamic logic and CMOS inverter stage is known as domino CMOS logic. Dynamic logic consists of a pull down network consists of nmos transistors, which implement the desired logic function. A clock signal is there for operation of the circuit. The circuit diagram is shown in Fig. 6.1.

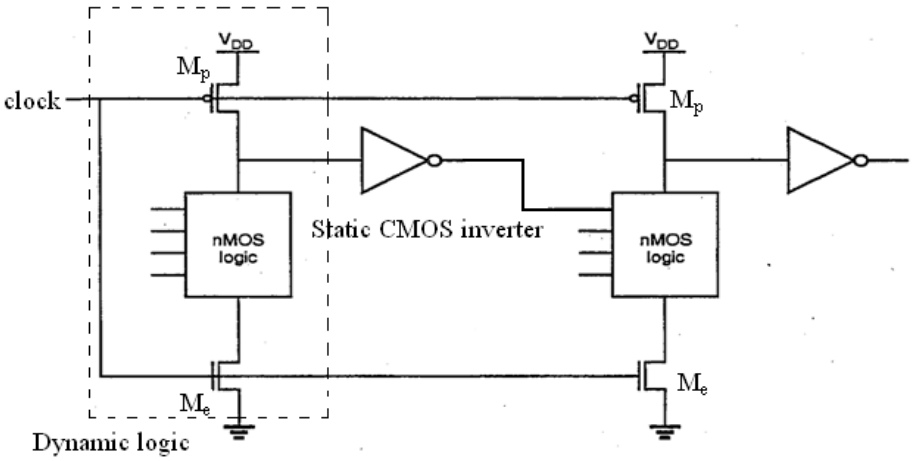

Fig. 6.1 Domino CMOS logic.

6.2.1 Dynamic Logic Operation

Operation of dynamic logic is divided into two stages, precharge and evaluation. When clock signal is 0, the circuit operates in precharged state. In this state the output is precharged to VDD value by pmos transistor, M_p. at that time evaluation nmos transistor M_e and pull down path is off. When clocl signal is 1, the circuit is in evaluation stage. During this period M_p is off and the output is evaluated according to the pull down network.

During evaluation stage output node can only discharges through ground node. Once the output is discharged it cannot be charged again until the next precharged operation. Thus the inputs to the gate can make only one transition during evaluation.

When multilevel logic structures are constructed by cascading dynamic logic circuit problem arises. To avoid the problem we are going for domino CMOS logic. In our modified adder structure due to the above reason domino CMOS logic structure is used.

6.2.2 Domino CMOS logic Operation

During precharge, output of n-type dynamic gate is charged upto Vdd and the output of inverter is set to 0. During evaluation dynamic gate conditionally discharges and output of inverter makes a conditional transition from 0 to 1. Only noninverting logic is implemented using domino logic as a static inverter is there at the output. High speed is achieved in this logic. When multi output block is implemented even number of static inverters are used to connect from one stage to the other to get correct output.

6.3 Modified Adder Structure

Different blocks of modified adder structure are described in following sub sections.

6.3.1 CLA0 Block

This block is a 4 bit carry look ahead adder taking Cin = 0. Transistor level schematic diagram is shown in Fig. 6.2. This block contains following sub blocks, which are described as follows.

Carry Propagator Generator Block:

This block generates Pi and Gi. Boolean equation for Pi and Gi are as follows:

Pi = Ai xor Bi

Gi = Ai and Bi

Where Ai, Bi are the two multi bit binary numbers to be added. This is done by instantiating 2 input XOR and AND gate. This block is shown in Fig. 6.3.

Carry Look Ahead Generator Block:

This block generates internal carry. Boolean equations for the internal carry are as follows:

C2 = G1 or (P1 and G0)

C3 = G2 or P2 (G1 or (P1 and G0))

C4 = G3 or P3 (G2 or P2 (G1 or (P1 and G0)))

This is done by instantiating 2 input XOR, AND and OR gates. This block is shown in Fig. 6.4.

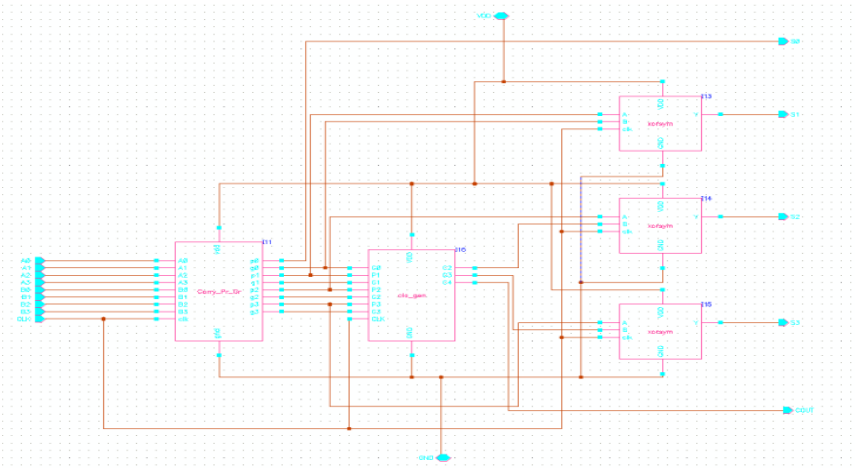

Fig. 6.2 Schematic of CLA0 block

Two Input XOR gate:

Three two input XOR gates are there to generate sum output of 4 bit adder. Boolean equations for sum outputs are:

$S_i = P_i$ xor C_i.

6.3.2 CLA1 Block

This block is a 4 bit carry look ahead adder taking $C_{in} = 1$. Transistor level schematic diagram is shown in Fig. 6.6. This block contains following sub blocks, which are described as follows.

Carry Propagator Generator Block:

This block generates P_i and G_i. Boolean equation for P_i and G_i are as follows:

$$P_i = A_i \text{ xor } B_i$$

$$G_i = A_i \text{ and } B_i$$

Where A_i, B_i are the two multi bit binary numbers to be added. This is done by instantiating 2 input XOR and AND gate. This block is shown in Fig. 6.3.

Carry Look Ahead Generator Block:

This block generates internal carry. Boolean equations for the internal carry are as follows:

$C_1 = G_0$ or P_0

$C_2 = G_1$ or P_1 and (G_0 or P_0)

$C_3 = G_2$ or P_2 (G_1 or P_1 and (G_0 or P_0))

$C_4 = G_3$ or P_3 (G_2 or P_2 (G_1 or P_1 and (G_0 or P_0)))

This is done by instantiating 2 input XOR, AND and OR gates. This block is shown in Fig. 6.5.

Two Input XOR gate:

Three two input XOR gates are there to generate sum output of 4 bit adder. Boolean equations for sum outputs are:

$S_i = P_i$ xor C_i.

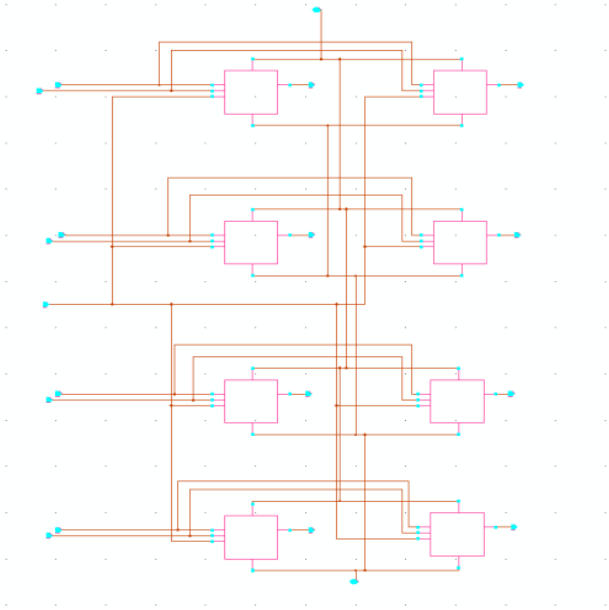

Fig. 6.3 Carry propagator Generator Block.

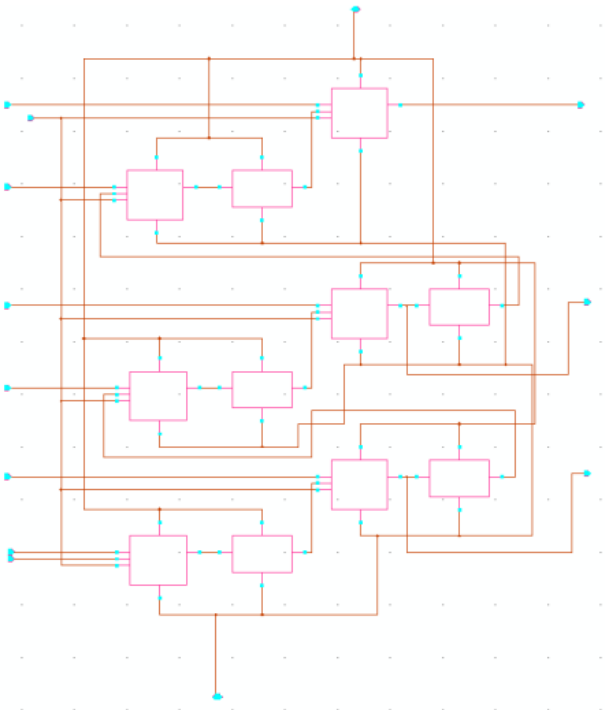

Fig. 6.4 Carry Look Ahead Generator Block used in CLA0 block.

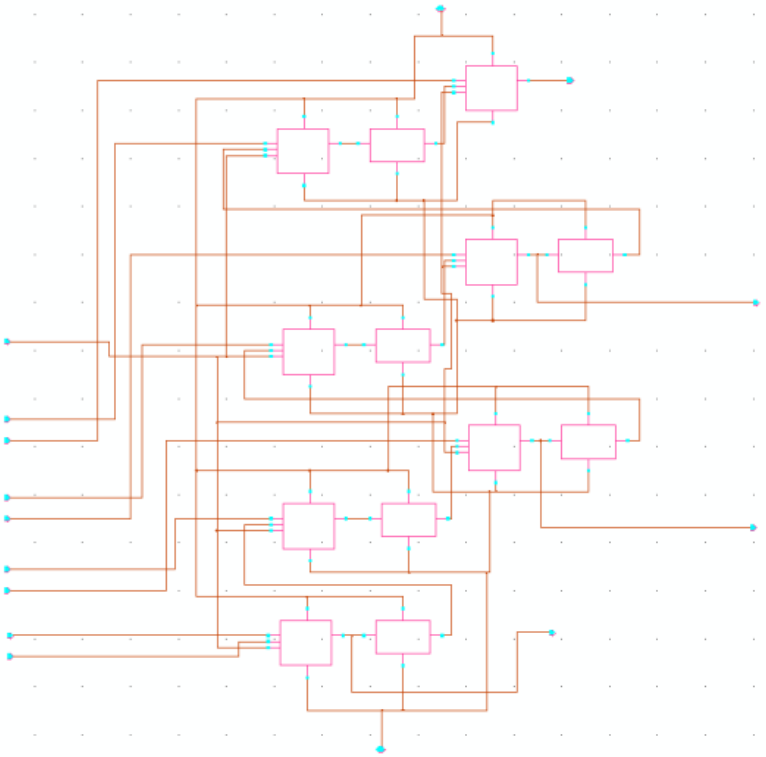

Fig. 6.5 Carry Look Ahead Generator Block used in CLA1 block.

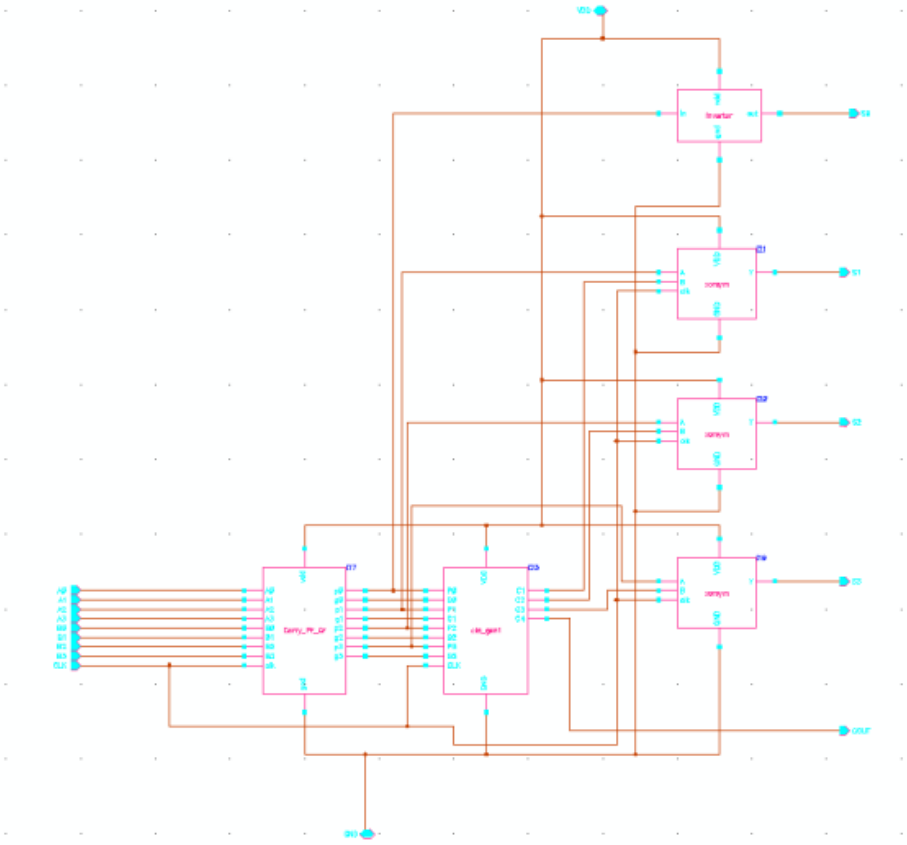

Fig. 6.6 Schematic of CLA0 block

6.3.3 Basic Gates used in Modified Adder Structure

This section shows transistor level diagram and layout diagram of basic gates such as CMOS inverter, buffer, 2 input AND gate, 2 input OR gate, 2 input XOR gate used in modified adder structure from Fig. 6.7 to 6.14.

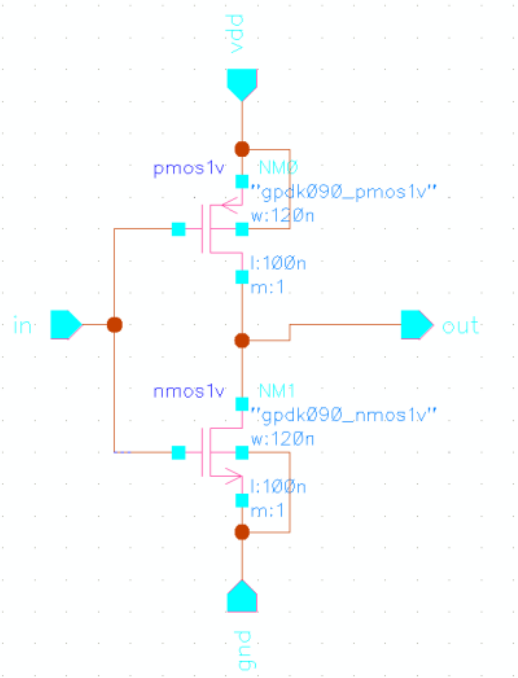

Fig. 6.7 Schematic of CMOS Inverter.

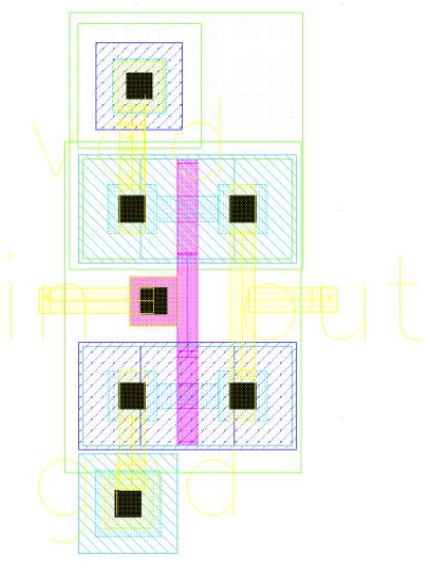

Fig. 6.8 Layout diagram of CMOS Inverter.

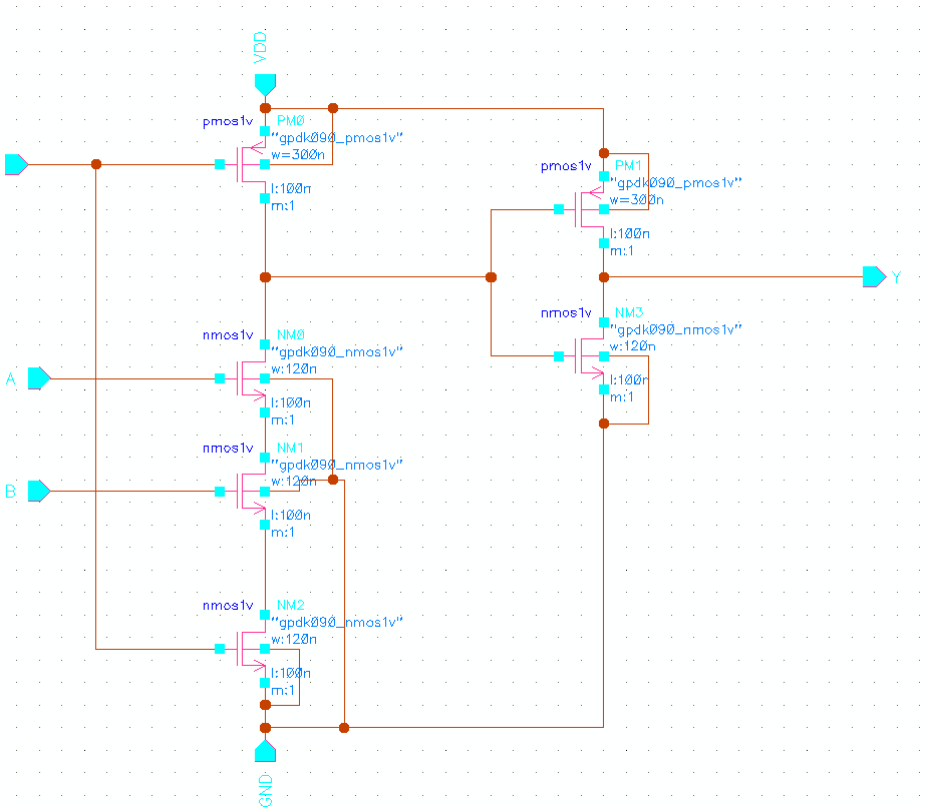

Fig. 6.9 Schematic of 2 input AND gate.

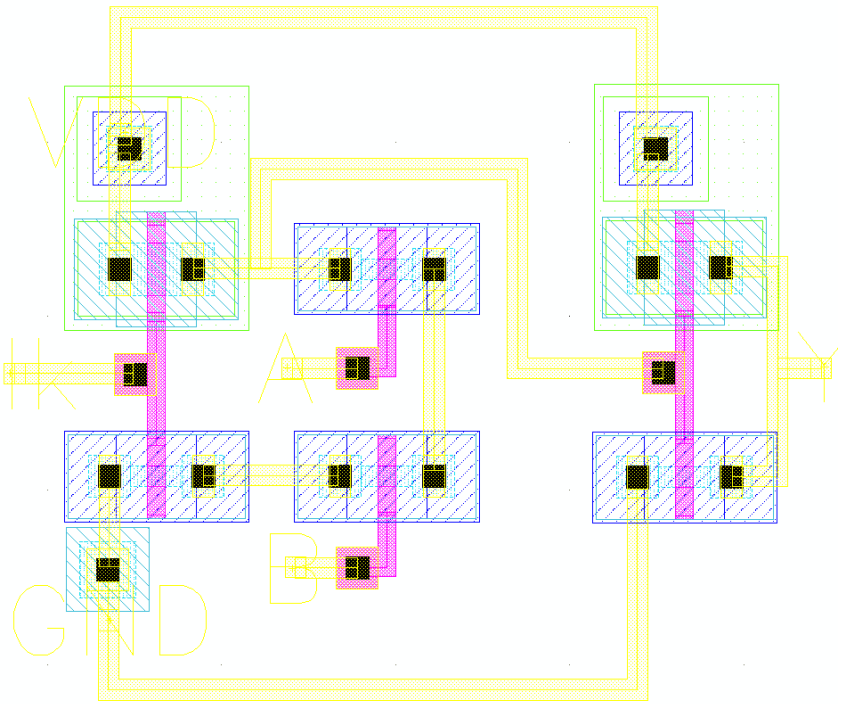

Fig. 6.10 Layout diagram of 2 input AND gate.

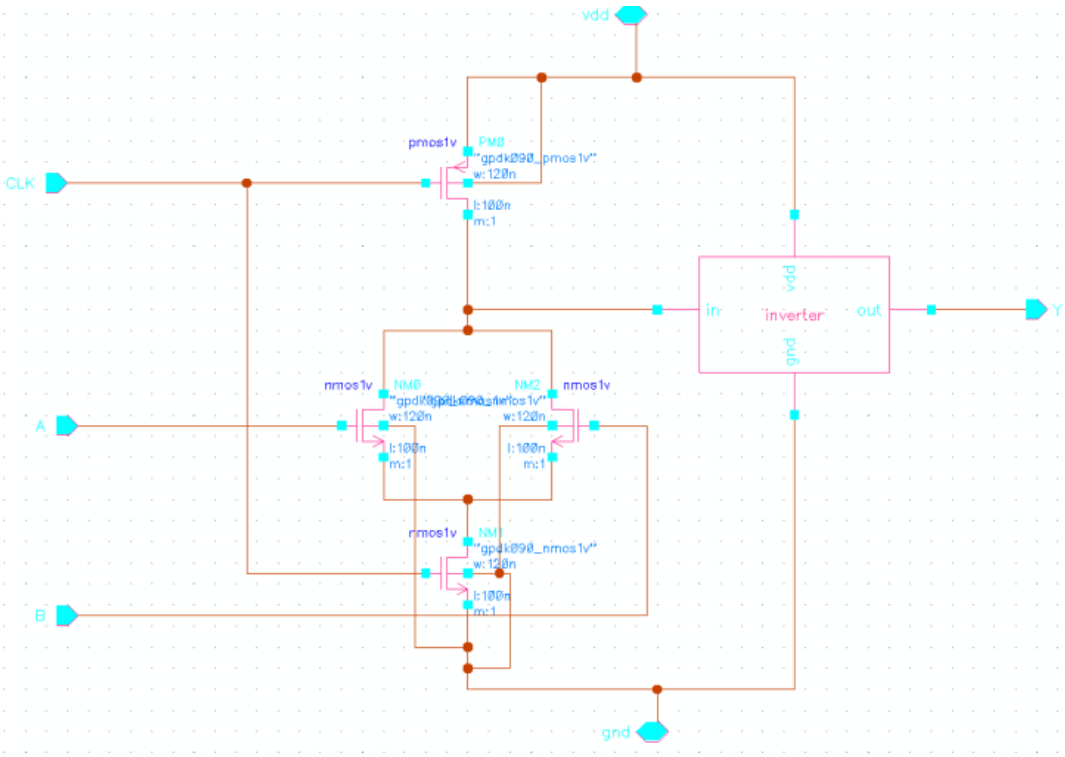

Fig. 6.11 Schematic of 2 input OR gate.

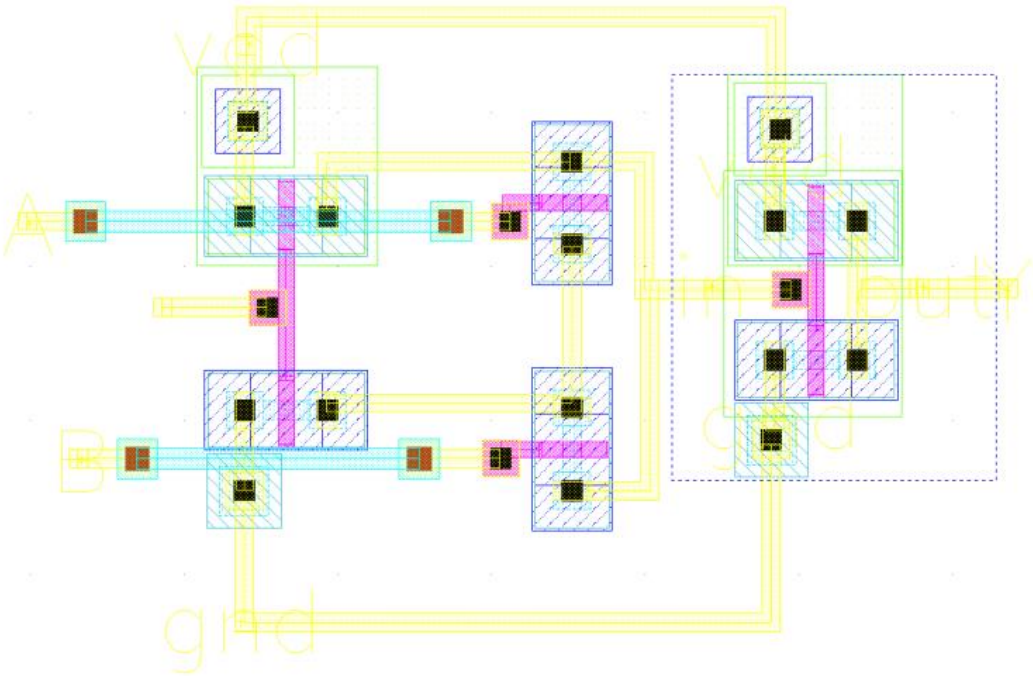

Fig. 6.12 Layout diagram of 2 input OR gate.

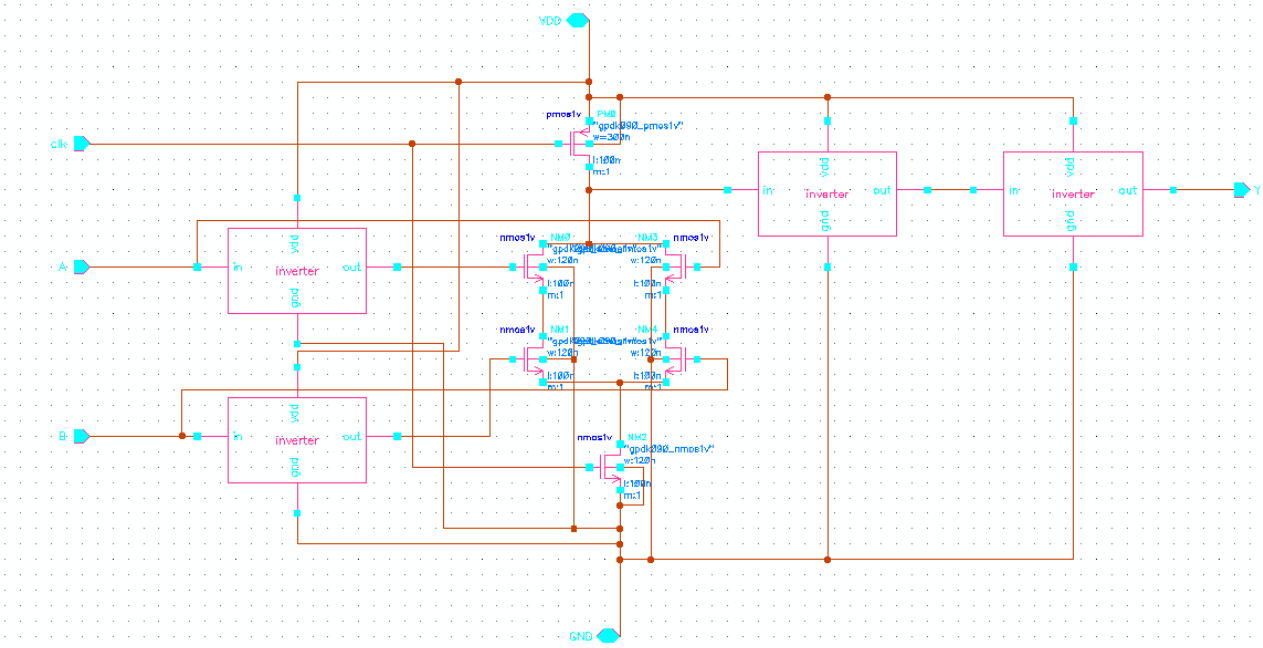

Fig. 6.13 Schematic of 2 input XOR gate.

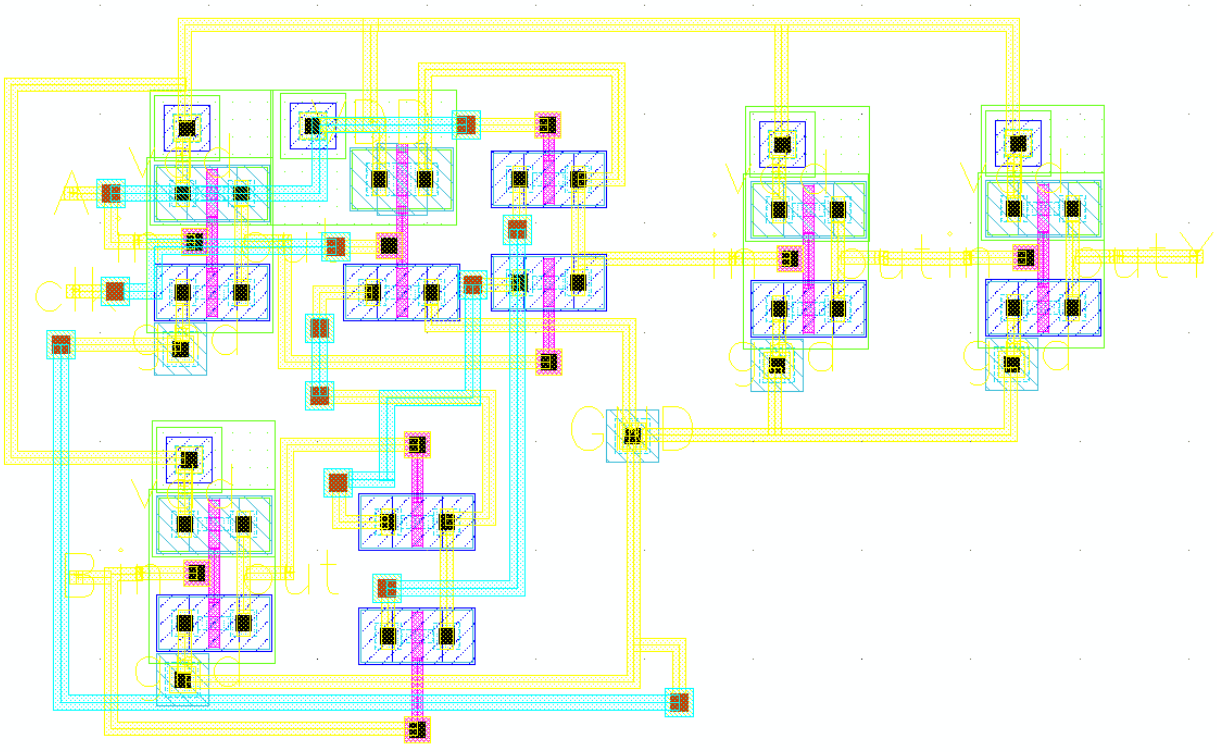

Fig. 6.14 Layout diagram of 2 input XOR gate.

6.4 Simulation Results

Transient analysis of each block is done in Cadence tool, version 5.1.41. simulation result for CLA0 and CLA1 block is shown in Fig. 6.15 and Fig. 6.16.

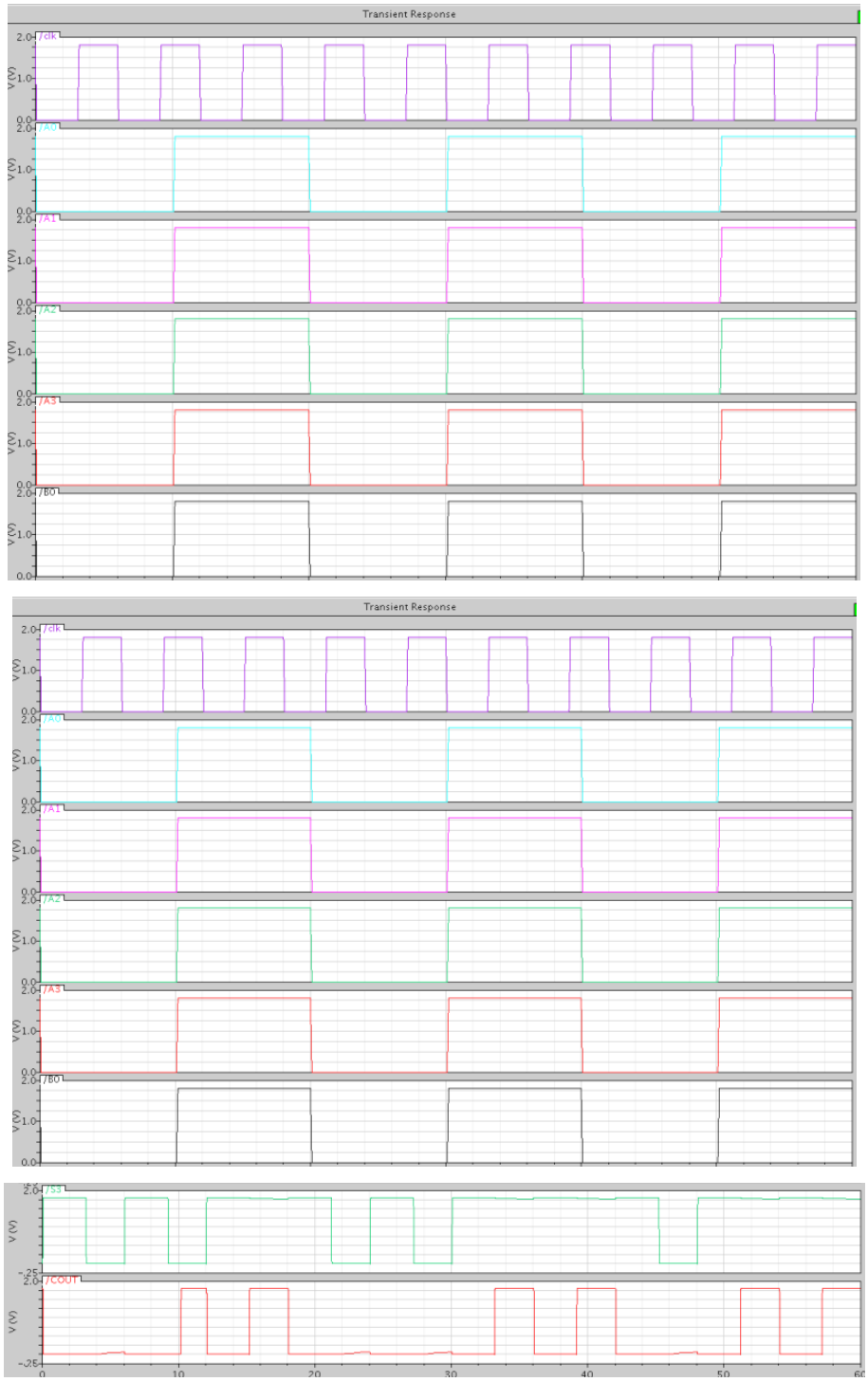

Fig. 6.15 Simulation result of CLA0 Block.

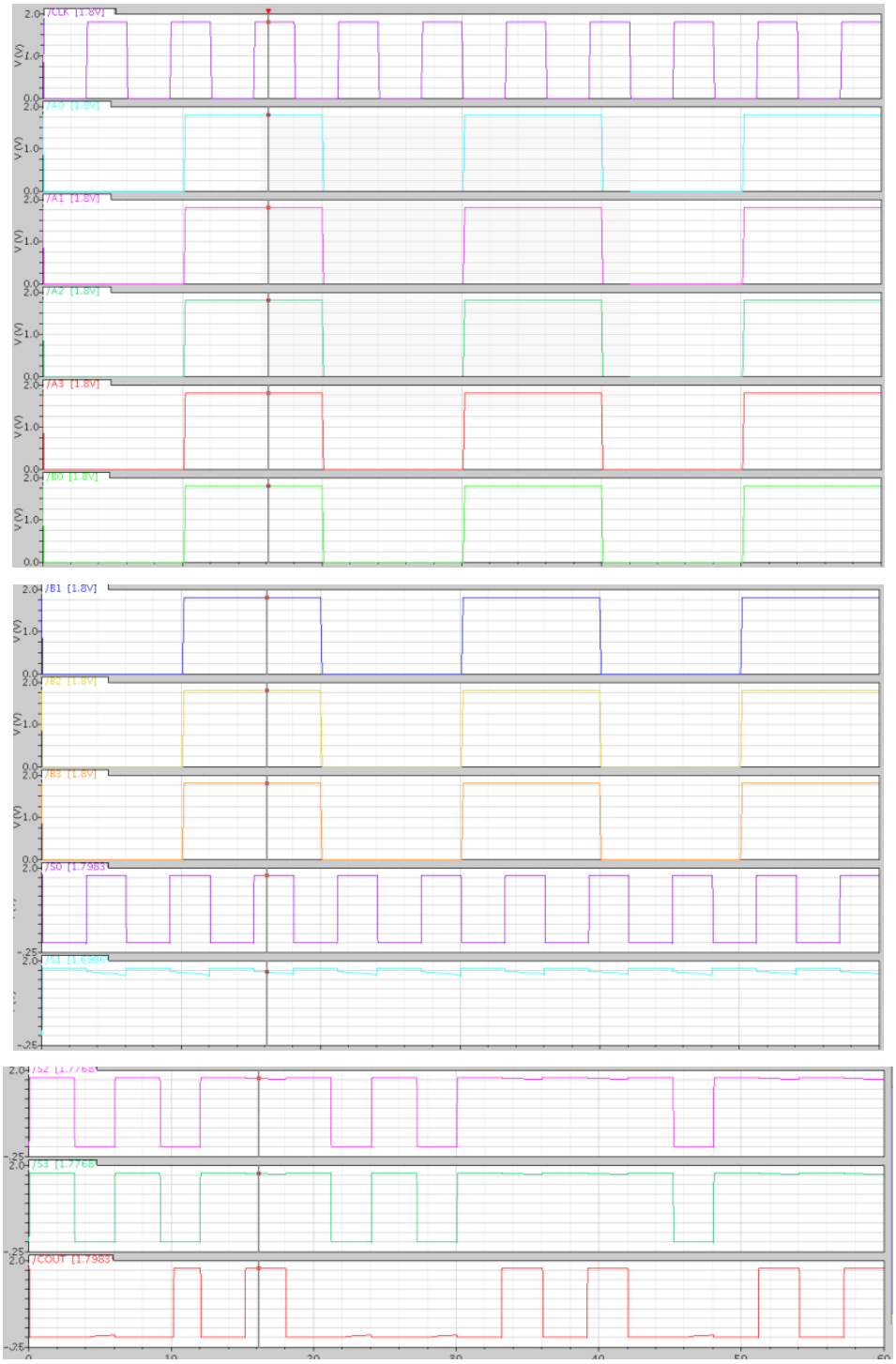

Fig. 6.16 Simulation result of CLA1 Block.

6.5 Conclusions

In this chapter transistor level implementation od modified adder structure is described. This was done using Cadence tool version 5.1.41. Schematic level design was done in 90nm technology. Layout diagrams are drawn taking layers of gpdk library. LVS, DRC, RC extraction and post layout simulations are done for basic gates. Simulation result of 4 bit CLA adder structure is shown.

Chapter 7 Conclusion and Future Work

7.1 Introduction

This thesis gives some idea about VLSI implementation of RSA cryptosystem. In this thesis basic computational blocks of RSA cryptosystem was described and implemented in VHDL, an HDL.

7.2 Our Contribution

Basics of cryptography and asymmetric key cryptography in particular were studied. RSA, an asymmetric key cryptosystem was studied in detail. VLSI implementation was done for computational block of encryption and decryption module of RSA cryptosystem. Computational block of RSA are modular multip[lier and modular exponentiation. Montgomery modular multiplication algorithm was studied. This algorithm was coded using VHDL to generate netlist for modular multiplication block using Xilinx ISE 10.1 environment. For modular exponentiation block MSB first algorithm was studied and implemented using VHDL.

Structure of modular multiplier was modified by taking a modified adder structure, which was a combination of CLA and CSL structure. Gate level and transistor level implementation was done for this adder structure using Xilinx and Cadence tool. From the result it can be concluded that this adder structure gives better delay performance as compared to other adder structure.

In this thesis we have tried to optimize the modular multiplier block, basic block of RSA cryptosystem. It was studied that throughput of modular multiplier can be increased in two ways, either we can decrease the computational cycle of it or we can reduce the critical path of it. In this work critical path was reduced by using a modified adder structure.

We can go for reduction of computation cycle of modular multiplier block in future.

Bibliography

[1]B. A. Forouzan, D. Mukhopadhyay, Cryptography and Network Security, Tata McGraw-Hill, 2012.

[2] W. Diffie and M. Hellman, "New directions in cryptography," IEEE Trans. Inf. Theory, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[3] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Commun. ACM, vol. 21, pp.120–126, Feb. 1978.

[4] J.-P. Deschamps, J. L. Imaña, and G. D. Sutter, Hardware Implementation of Finite-Field Arithmetic. New York: McGraw-Hill, 2009, ser. Electronic engineering series.

[5] P. L. Montgomery, "Modular multiplication without trial division, "Math. Comput., vol. 44, no. 170, pp. 519–521, Apr. 1985. [Online]. Available: http://www.jstor.org/stable/2007970.

[6] Atsushi Miyamoto, Naofumi Homma, et al."Systematic Design of RSA Processors Based on High-Radix Montgomery Multipliers," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 19, No. 7, July 2011.

[7] Jun-Hong Chen, Ming-Der Shieh, "A High-Performance Unified-Field Reconfigurable Cryptographic Processor," IEEE Trans. on Very Large Scale Integration (VLSI) systems, vol. 18, no. 8, August 2010.

[8] A. P. Fournaris and O. Koufopavlou, "A new RSA encryption architecture and hardware implementation based on optimized Montgomery multiplication," in *Proc. IEEE ISCAS*, May 23–26, 2005, pp. 4645–4648.

[9] S. B. Ors, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of a Montgomery modular multiplier in a systolic array," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2003, p. 185.1.

[10] Gustavo D. Sutter, Jean-Pierre Deschamps, and José Luis Imaña, "Modular Multiplication and Exponentiation Architectures for Fast RSA Cryptosystem Based on Digit Serial Computation," IEEE Transactions on Industrial Electronics, Vol. 58, No. 7, July 2011.

[11] Kiat – Seng Yeo, Kaushik Roy, Low- Voltage, Low-Power VLSI Subsystems, Tata McGraw-Hill, 2009.

[12] O.J. Bedriji, "Carry select adder," IRE Trans. Electronics Computers, pp. 340-346, Sept. 1961.

[13] J. M. Rabaey et al, Digital Intigrated Circuits- A Design Perspective, Pearson Education.

[14] Kang, Leblebici, CMOS Digital Integrated Circuits – Analysis and Design, Tata McGraw-Hill, 2003.

[15] D. L. Perry, VHDL: Programming by Example, Tata McGraw-Hill.

[16] Xilinx ISE Design Suite 10.1 Software Manuals, 10th ed., Xilinx Inc., San Jose, CA, 2008.